Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Glumac

Sistem za beleženje športne aktivnosti

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Robert Rozman

Ljubljana, 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Sistem za beleženje športne aktivnosti

Tematika naloge:

Beleženje športne oziroma gibalne aktivnosti je trenutno zelo zanimivo področje, ki je pritegnilo tudi največja podjetja s področja informacijsko-komunikacijskih tehnologij (IKT); vendar se njihovi pristopi in rešitve zelo razlikujejo. Še vedno ni povsem jasno kakšna bo rešitev, ki bo prevladala na tem področju. Po lastni ideji razvijte in implementirajte prototip sistema za beleženje športne aktivnosti z vsem potrebnim za samostojno delovanje. Sistem naj bo sestavljen iz mobilnega dela, ki ga uporabnik nosi pri športni aktivnosti in strežniškega dela, kjer se shranjujejo podatki o izvedenih športnih aktivnostih. Prav tako naj strežniški del poskrbi za primerno prezentacijo shranjenih podatkov uporabniku.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matjaž Glumac sem avtor diplomskega dela z naslovom:

Sistem za beleženje športne aktivnosti

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Roberta Rozmana ,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 14. septembra 2015 Podpis avtorja:

Zahvaljujem se mentorju viš. pred. dr. Robertu Rozmanu, za pomoč, strpnost in obilo prijaznosti pri izdelavi diplomskega dela.

Diplomsko delo posvečam svoji družini, ki mi je pomagala finančno in psihično skozi leta študija, ter dekletu Tei, ki mi je dala navdih za nove cilje.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
GPS	Global positioning system	globalni navigacijski satelitski sistem
LED	Light-emitting diode	svetleča dioda
IDE	integrated development environment	integrirano razvojno okolje
OS	Operating system	operacijski sistem
HDMI	High-definition multimedia interface	multimedijski vmesnik visoke ločljivosti
ARM	Advanced reduced instruction set computer machine	napreden računalnik z zmanjšano množico ukazov
MHz	Megahertz	megaherc
GB	Gigabyte	gigabajt
RAM	Random access memory	bralno-pisalni pomnilnik
USB	Universal serial bus	univerzalno serijsko vodilo
GPIO	General-purpose input/output	splošno namenski vhodi/izhodi
NPM	Node package manager	nadzornik paketov okolja NodeJS
DDV	Value added tax	davek na dodano vrednost
JSON	JavaScript Object Notation	oblika zapisa podatkov, ki se uporablja pri spletnih tehnologijah
I2C	Inter-integrated circuit	povezava med integriranimi vezji
ATTRS	Attributes	lastnosti

SPI	Serial Peripheral Interface Bus	standard za sinhrono serijsko podatkovno povezavo
API	Application programming interface	aplikacijski programski vmesnik
V	Volt	merska enota za merjenje napetosti
BPS	Bits per second	biti na sekundo
SDA	Data line	podatkovna povezava
SCL	Clock line	povezava z urinim signalom
A-GPS	Assisted global positioning system	pomožni globalni navigacijski satelitski sistem
ICSP	In-Circuit Serial Programming	programiranje vgrajenih naprav, mikrokrmilnikov
KB	Kilobyte	kilobajt
SRAM	Static random access memory	dinamični bralno-pisalni pomnilnik
EEPROM	Electrically Erasable Programmable Read-Only Memory	električno izbrisljiv programirljiv bralni pomnilnik
AC	Alternating current	izmeničen tok
DC	Direct current	enosmerni tok
MEAN	MongoDB ExpressJS AngularJS NodeJS	zbirka tehnologij JavaScript za razvoj spletnih aplikacij
DB	Database	podatkovna baza
JS	JavaScript	skriptni jezik JavaScript
MVC	Model-View-Controller	načelo za razvijanje spletnih vsebin, model-pogled-kontrola
HTML	HyperText Markup Language	jezik za pisanje spletnih strani
URL	Uniform Resource Locator	enolični krajevnik vira
MET	Metabolic equivalent of task	merska enota za porabo energije pri fizični aktivnosti

Povzetek

Ideja diplomskega dela je sistem za shranitev in prikazovanje podatkov športne aktivnosti. Sistem je sestavljen iz prenosnega in zalednega podsistema. Prenosni podsistem pridobiva informacije o lokaciji, hitrosti, vibracijah med vadbo in razdalji, opravljeni med vadbo. Oseba si prenosni podsistem spravi v torbico za okrog pasu. Med vadbo se informacije zbirajo. Ko oseba zaključi vadbo priključi prenosni podsistem na zaledni podsistem. Podatki se takoj samodejno začnejo prenašati med podsistemoma. Zaledni podsistem skrbi za shranitev podatkov o posameznih vadbah in skrbi za njihov spletni prikaz. V primerjavi z obstoječimi rešitvami naš sistem prikazuje tudi vibracije med našo vadbo, ki jih je mogoče primerjati z referenčno vrednostjo.

Ključne besede: diplomsko delo, Arduino, Raspberry Pi, tek, vadba.

Abstract

The idea behind the thesis is a system for storing and web presentation of sports activity data. The system consists of a portable and a back-end subsystem. The portable subsystem acquires data about the location, speed, vibrations during exercise and the distance covered. The person keeps the portable subsystem in a case around the waist. Data is collected during exercise. As soon as the person completes the session, the portable system can be connected to the back-end subsystem. The two systems immediately start transmitting data between them. The back-end subsystem stores the data about individual sessions and enables their presentation on the internet. Compared to the existing solutions, our system can display vibrations during exercise, which can be compared to a reference value.

Keywords: thesis, Arduino, Raspberry Pi, running, exercise.

Poglavje 1

Uvod

Danes imamo veliko izbiro naprav, ki omogočajo beleženje in prikazovanje podatkov med športno aktivnostjo. Imamo pametne ure, zapestnice, telefone, GPS-sprejemnike, itd. Večina teh naprav prikazuje podatke, ki jih ne potrebujemo oz. jih ne želimo uporabiti ali pa jih ne razumemo.

Diplomsko delo prikazuje izdelek, ki vsebuje specifične funkcije, ki jih najpogosteje potrebujejo rekreativni in amaterski tekači. Avtor diplomskega dela je motivacijo za izdelek dobil, ker je tudi sam športnik. Pripomočki, ki jih uporablja za meritev treningov, mu ne ponujajo dovolj preproste rešitve. Poleg tega ne nudijo nobene pomoči ljudem, ki imajo probleme s poškodbami. Za ta namen smo hoteli v izdelek implementirati rešitev, ki bi prikazovala obremenjenost telesa med rekreacijo.

Ideja diplomskega dela je izdelava celotnega sistema, ki je sestavljen iz dveh podsistemov:

- prenosni podsistem za beleženje aktivnosti s tipali
- zaledni podsistem za shranitev in prikaz podatkov

Prenosni podsistem vsebuje napravo Arduino Uno R3, na kateri so priključena elektronska tipala. Ta tipala skrbijo za zajem podatkov. Podatki, ki jih tipala zajamejo, so:

- vibracije zaznane med vadbo

- hitrost osebe
- lokacije, na katerih se nahaja oseba med treningom
- razdalja, ki jo je opravila oseba med rekreacijo

Zaledni podsistem skrbi, da se podatki po končani vadbi shranijo v podatkovno bazo in se prikazujejo na spletni strani. Omenjeni podsistem je realiziran s pomočjo miniaturnega računalnika Raspberry Pi 2, model B.

Na začetku smo se osredotočili le na izdelavo podsistemov, ki so nujni za delovanje celotnega sistema. Ko smo celoten sistem razvili, smo opazili, da bi bilo mogoče v prihodnosti dopolniti sistem z dodatnim podsistemom, ki bi omogočal priključitev več različnih prenosnih podsistemov na isti zaledni podsistem.

V naslednjem poglavju bomo predstavili strukturo sistema. Omenili bomo tudi orodja in tehnologije, ki so bile uporabljene pri razvoju. V tretjem poglavju, bo predstavljen zaledni podsistem. Četrto poglavje je namenjeno prenosnemu podsistemu, v katerem smo opisali elektronska tipala, ki smo jih uporabili. V petem poglavju je opis delovanja in strukture spletne strani, na kateri dostopamo do podatkov, ki smo jih zajeli med vadbo. Predzadnje poglavje vsebuje opis delovanja celotnega sistema diplomske naloge. Zadnje poglavje obsega predstavitev sklepnih ugotovitev.

Poglavje 2

Zgradba sistema

Celotno diplomsko delo je delujoči sistem, ki nam omogoča pridobivanje podatkov med rekreacijo in tudi prikaz teh podatkov na internetu. Celoten sistem je razdeljen na dva podsistema:

- prenosni podsistem
- zaledni podsistem

Prenosni podsistem je sestavljen iz naprave Arduino Uno R3 in elektronskih tipal, s katerimi zajamemo podatke. Uporabljena tipala so:

- GPS-modul GY-GPS6MV2
- modul za spominsko kartico MicroSD Card Adapter
- pospeškometer, barometer in giroskop LSM9DS0 9DOF
- pretvornik napetosti iz 3,3 v 5 V in obratno

Za izvedbo in lažje razumevanje delovanja prenosnega podsistema imamo na napravi Arduino priključene fizične tipke in LED-diode.

Pri pisanju programske kode prenosnega podsistema smo uporabljali programsko okolje Arduino IDE [1]. Okolje izvira iz projekta Processing [8].

Omenjeno okolje podpira pisanje programov v C++, C ali Arduino programskih jezikih. V našem primeru je vsa programska koda, ki je napisana za prenosni podsistem, napisana v programskem jeziku Arduino.

Zaledni podsistem je implementiran s pomočjo miniaturnega računalnika Raspberry Pi 2, model B. Njegovi nalogi sta beleženje in prikaz podatkov, ter prenos podatkov iz prenosnega podsistema.

Za realizacijo zalednega podsistema, je bilo potrebno:

- namestiti bazo NoSQL MongoDB [4] na Raspberry Pi 2 , model B
- namestiti izvajalno okolje NodeJS [5] za gostitev spletne strani
- sprogramirati spletno stran v ogrodju AngularJS [2] za JavaScript
- napisati program v programskem jeziku Python, ki je prenesel podatke iz prenosnega podsistema in jih shranil v podatkovno bazo
- preurediti nastavitve sistemskih datotek Raspberry Pi, za avtomatski zagon programa za prenos podatkov ob priklopu prenosnega podsistema nanj

Razvojno okolje, uporabljeno za razvoj zalednega podsistema, je bilo Visual Studio Code, izdelek Microsoft-a [10]. Njegova velika prednost je, da ga lahko namestimo na vse moderne operacijske sisteme (Windows, Linux, Mac).

Na zalednem podsistemu uporabljamo operacijski sistem Raspbian [6], ki je pomanjšana in prilagojena oblika operacijskega sistema Debian [7] za računalnike Raspberry Pi.

Ker je celotni sistem razdeljen na dva dela, nastane problem pri hranjenju datotek. Zaradi omenjenega problema se je uporabilo sistem Git. To je sistem za nadzor nad porazdeljenimi revizijami projekta. Danes je to zelo koristno v odprtokodnih projektih ali velikih organizacijah, kjer dostopa do projekta veliko število oseb. Sistem v osnovi deluje tako, da imamo osnovno različico projekta na glavni veji sistema, ki jo razdelimo v različne podveje.

V podvejah lahko spreminjamo projekt, čeprav se to na glavni veji ne bo poznalo. Ko enkrat zaključimo dopolnjevanje projekta na podveji, združimo novo različico projekta iz podveje skupaj z glavno vejo. Če želimo, lahko projekt naložimo v spletno shrambo (oblak-GitHub, Bitbucket) ali ohranimo le lokalno na računalniku.

Poglavje 3

Zaledni podsistem za shranitev in prikaz podatkov

Vloga zalednega podsistema je, da hrani podatke prenesene iz prenosnega podsistema in poskrbi za prikaz teh podatkov na spletni strani. Zaledni podsistem deluje na računalniku Raspberry Pi 2, model B. Razlog za uporabo računalnika Raspberry Pi 2, model B, je v tem, da je računalnik majhen, tih in neopazen v prostoru. Poleg omenjenih lasnosti je prednost tudi v tem, da je na spletu veliko pomoči za začetnike.

V naslednjem podpoglavju je predstavljen računalnik Raspberry Pi 2, model B. Sledila bodo podpoglavja z opisom tehnologije, uporabljene za realizacijo omenjenega podsistema.

3.1 Implementacija zalednega podsistema

Za implementacijo zalednega podsistema smo uporabili računalnik Raspberry Pi 2, model B. To je računalnik velikosti kreditne kartice, ki ga je mogoče kupiti od februarja 2015 naprej. V času pisanja diplomskega dela je ta izdelek najnovejši izdelek iz družine Raspberry Pi.

Prvi tak računalnik v družini Raspberry Pi, je bil Raspberry Pi, model A. Proti najnovejšemu je precej počasnejši (več kot 15-krat počasnejši). Njegov

Slika 3.1: Raspberry Pi 2, model B [16]

namen je bil, da bi otroke v šoli lahko učili programiranja. Danes se omenjeni računalnik velikokrat uporablja pri realizaciji projektov.

Raspberry Pi 2, model B, ima sledeče specifikacije:

- priključek HDMI za monitor
- štirijedrni procesor ARM7 z 900MHz na posamično jedro
- 1 GB RAM
- 4 priključki USB
- mikro priključek USB za napajanje
- priključek za kartico MicroSD
- 40 razširitvenih pinov GPIO
- priključek Ethernet
- vhodni priključek za standardni avdio priključek premera 3,5 milimetra, ali analogni video priključek

Specifikacije, navedene zgoraj, omogočajo realizacijo različnih projektov. Nekateri si z njim naredijo nadzorno kamero pred vhodnimi vrati, katere sliko lahko vidijo neposredno na internetu. Drugi si naredijo svoj radio in oddajajo glasbo na internet. Nekateri si namestijo poseben operacijski sistem, ki spremeni omenjeni računalnik v medijski center.

Omogoča nam priključitev tipal, kot so:

- kamera
- senzor vlage in temperature
- senzor oddaljenosti
- senzor gibanja

Vsa tipala lahko priključimo na izbrane priključke izmed štiridesetih mogočih priključkov. Njihovo delovanje kontroliramo s programsko kodo, ki jo napišemo v željenem programskem jeziku. Seveda moramo imeti za izbrani programski jezik ustrezne prevajalnike ali interpreterje programske kode. Ker je največ primerov in programskih knjižnic spisanih v programskem jeziku Python, bi raje ostali le pri omembi Pythona.

Raspberry Pi 2, model B, je računalnik, na katerega lahko namestimo več različnih operacijskih sistemov. Seveda morajo biti operacijski sistemi prilagojeni za računalnik Raspberry Pi. Ravno zaradi večje zmogljivosti najnovejšega izdelka je mogoče namestiti operacijske sisteme, ki so bili nekoč prezahtevni. Za lažjo predstavo o tem so na sliki 3.2 specifikacije prvega, zadnjega in predzadnjega računalnika Raspberry Pi.

Kot je iz slike 3.2 razvidno, se vsi modeli razlikujejo le po velikosti pomnilnika, številu priključkov USB in številu razširitvenih priključkov. Izjema je zadnji model, ki ima štirijedrni procesor. Razlog, da so vse karakteristike skoraj enake, je ta, da lahko uporabniki ohranijo kompatibilnost projektov za naprej in nazaj. Razlika, ki jo je treba omeniti je tudi ta, da prva različica nima priključka Ethernet. Če ga potrebujemo, pa je mogoče dokupiti dodatni priključek ki se ga priključi na Raspberry Pi.

Ime modela	Pomnilnik RAM (MB)	Procesor	Video	Avdio	Razširitveni pini	Ethernet priključek	USB priključeki
Raspberry Pi model A	256	Arm7 700MHz	HDMI in analogni video priključek	3.5 avdio priključek	26	Ne	1
Raspberry Pi model B+	512	Arm7 700MHz	HDMI in analogni video priključek	3.5 avdio priključek	40	Da	4
Raspberry Pi 2 Model B	1024	Arm7 700MHz, štiri jedra	HDMI in analogni video priključek	3.5 avdio priključek	40	Da	4

Slika 3.2: Primerjava modelov Raspberry Pi

Spodaj so omenjeni operacijski sistemi, ki jih je trenutno oz. v času pisanja diplomskega dela mogoče poganjati na Raspberry Pi 2, model B:

- Raspbian (vsi modeli)
- Ubuntu Mate (le najnovejši)
- Snappy Ubuntu core (le najnovejši)
- OSMC (vsi modeli)
- Openelec (vsi modeli)
- Pinet (vsi modeli)
- Risc OS (vsi modeli)
- Windows 10 (le najnovejši)
- Arch Linux ARM (vsi modeli)

V napravi Raspberry Pi 2, model B, iz diplomskega dela je nameščen operacijski sistem Raspbian. Razlog, da smo se odločili za uporabo tega sistema, je, da se omenjeni operacijski sistem uporablja že od prvega modela. Ravno zaradi tega je veliko programov za omenjeni operacijski sistem, ki nam lajšajo delo.

3.2 Delovanje zalednega podsistema

Na Raspberry Pi deluje zaledni podsistem, ki je razvit z ogrodjem ExpressJS [3] za izvajalno okolje NodeJS [5]. Okolje NodeJS nam omogoča izdelovanje in poganjanje omrežnih aplikacij. Aplikacije se razvija v programskem jeziku JavaScript. V diplomskem delu se je izvajalno okolje NodeJS uporabljalo za izdelavo in izvajanje zalednega podsistema. Za pravilno izvajanje okolja NodeJS je treba najprej namestiti orodje `Node package manager` (v nadaljevanju NPM).

NPM je orodje za nadzor knjižnic in modulov JavaScript. Z namestitvijo NPM na naš računalnik se avtomatsko že namesti izvajalno okolje NodeJS. Prednost uporabe orodja NPM je preprost način nameščanja knjižnic. Z enim ukazom se nam prenese knjižnica s spleta v direktorij, v katerem imamo zaledni podsistem.

```
$ npm install moduleName
```

Slika 3.3: Primer ukaza z orodjem NPM

Primer na sliki 3.3 prikazuje način, kako v terminalu na operacijskem sistemu Linux namestimo določen modul ali knjižnico. Za uporabnike operacijskih sistemov Windows je ta ukaz enak, le način namestitve orodja NPM je drugačen.

Zaledni podsistem skrbi za avtentikacijo uporabnika, shranitev podatkov v bazo in prikaz podatkov. Trenutno nima prav veliko nalog, vendar nam lahko v primeru nadaljnjega razvoja omogoči obdelavo podatkov v ozadju.

3.2.1 Ogrodje ExpressJS za izvajalno okolje NodeJS

Izvajalno okolje NodeJS omogoča, da razvijemo zaledni podsistem na način, ki ga podpira omenjeno okolje. Ker bi za razvoj celotnega podsistema potrebovali veliko dodatne programske kode, smo raje uporabili ogrodje ExpressJS. To ogrodje si lahko predstavljamo kot nek ovoj celega izvajalnega okolja, ki ima to prednost, da namesto nas izvede postopke, ki bi jih morali sicer sami sprogramirati.

V ogrodju ExpressJS lahko dodajamo že prej ustvarjene module ali knjižnice, če pa imamo svoje module, jih lahko večkrat ponovno uporabimo. Dobra stran ogrodja ExpressJS je v tem, da lahko programiramo v programskem jeziku JavaScript, saj je spletna stran razvita z ogrodjem AngularJS za programski jezik JavaScript. To pomeni, da nam ni treba znati uporabljati dveh različnih jezikov, poleg tega pa lahko določene komponente iz spletne strani uporabimo tudi na zalednem podsistemu.

```
1 (function(){
2         var connection                = require('./auth.connection').connectToDB,
3         schemeUser                    = require('../schemes/schemes.user').user,
4         model                         = connection.model('users',schemeUser,'users'),
5         forge                         = require('node-forge'),
6         encrypt                       = require('./auth.cipher').encrypt,
7         messages                      = require('../auth/auth.messages');
```

Slika 3.4: Uvažanje modulov ExpressJS v programsko kodo

Na sliki 3.4 je primer uvoza obstoječih modulov v določen del programske kode. To storimo z rezervirano funkcijo `require(imeModula | potDoModula)`, v katero navedemo ime ali pot modula. Če imamo modul nameščen s spleta, potem navedemo le ime modula. V nasprotnem primeru navedemo pot kjer se nahaja modul, ki smo ga sami napisali. Tu se izrazi dejstvo, da sta spletna stran in zaledni podsistem razvita s programskim jezikom JavaScript. To pomeni, da lahko module, uporabljene v zalednem podsistemu, dodamo tudi na spletno stran, če bi to želeli.

3.2.2 Podatkovna baza MongoDB

Že mnogo let na vrhu uporabe podatkovnih baz vztrajajo relacijske podatkovne baze, kot so:

- MySQL
- MSSQL
- SQLite

Omenjene baze uporabljamo že toliko časa, da je vsak, ki je moral kdajkoli v projektu uporabiti bazo, skoraj zagotovo uporabil relacijsko bazo. Omenjene podatkovne baze so namreč razumljive in se štejejo za edine uporabne baze. Na sliki 3.5 je prikaz stavka, s katerim iz podatkovne baze SQL izberemo zapise z določenim datumom.

```
SELECT * FROM PRACTICE WHERE PRACTICE.date = '23-05-2015';
```

Slika 3.5: Primer stavka SQL

V zadnjem času se veliko uporabljajo baze NoSQL. Njihova največja prednost je ta, da se v en zapis shrani ogromno podatkov. Popularnost teh podatkovnih baz so dvignila socialna omrežja. Dober primer so komentarji na socialnem omrežju Facebook. Oseba ima lahko tudi več tisoč komentarjev. Namesto da baza poišče tisoč in več zapisov, se iz baze vrne le en zapis, ki hrani vse potrebne podatke. Na sliki 3.6 je prikaz stavka v bazi NoSQL, ki išče z enakim pogojem kot stavek SQL na sliki 3.5.

```
db.practice.find({"date":"23-08-2015"});
```

Slika 3.6: Primer stavka podatkovne baze MongoDB, ki išče z enakim pogojem kot stavek SQL na sliki 3.5.

Relacijske baze uporabljajo relacije za shranitev podatkov. Relacije so dejansko tabele, ki so razdeljene na stolpce in vrstice. Za podatke, ki jih je

možno razporediti v isto skupino, je taka vrsta baze odlična. Dober primer bi lahko bil račun, ki ga trgovina izstavi ob nakupu, ki vsebuje naslednje podatke:

- naziv trgovine
- naslov trgovine
- kontaktna številka
- datum in čas nakupa
- številka računa
- status računa
- seznam z nazivi artiklov, količino in njihovo ceno
- znesek nakupa brez DDV
- vrednost DDV
- skupni znesek z DDV
- ime in priimek blagajnika

Seznam podatkov je relativno majhen. Iz napisanega seznama je mogoče takoj ustvariti tabelo, v kateri bi v vsak stolpec vnesli svoj podatek. S tem bi vsaka vrstica predstavljala posamičen račun.

Tabela na sliki 3.7 ne ustreza popolnoma seznamu podatkov. Manjka seznam artiklov. Tabela je namenjena le za lažje razumevanje. Da bi bilo tehnično pravilno bi bilo treba uporabiti več različnih tabel, ki bi predstavljale račun, izdelek in seznam izdelkov. Tabele bi se med seboj povezovale preko enega skupnega podatka.

Na drugi strani imamo baze NoSQL, ki so uporabljene v tem diplomskem delu z implementacijo podatkovne baze MongoDB. V teh bazah ni tabel, pač pa so t.i. dokumenti, v katerih se hrani struktura podatkov parov ključa-vrednosti, imenovana JSON.

Naziv trgovine	Naslov trgovine	Kontaktna številka	Datum in čas nakupa	Številka računa	Status računa	Znesek nakupa brez DDV(EUR)	Vrednost DDV(EUR)	Skupni znesek z DDV(EUR)	Ime in priimek blagajnika
Market rožna dolina Janez Novak	Rožna dolina, c. IX 17, LJUBLJANA	051284325	07.03.2013:19:41	160/02012889	Potrjeno	9	1,50	10,50	

Slika 3.7: Primer računa, predstavljenega z relacijo

JSON ali `JavaScript Object Notation` je format, ki uporablja človeško berljivo besedilo za prenos podatkovnih objektov, ki so sestavljeni iz ključa-vrednosti parov. Čeprav je izpeljan iz programskega jezika JavaScript, je format JSON neodvisen od programskega jezika. Pretvarjati in uporabljati tak format je danes mogoče že v mnogih programskih jezikih. Točno to predstavlja strukturo podatkov v dokumentu. Če bi zgornji primer iz relacijske baze postavili v obliko za MongoDB, bi dobili dokument, ki ga predstavlja slika 3.8.

Na sliki 3.8 je mogoče videti celoten seznam podatkov, vključno s seznamom artiklov. Zelo preprosto je neko strukturo ponazoriti v objektu JSON. Glede na sliko bi vsakdo takoj predpostavil, da je uporaba take baze preprosta in hitrejša, ampak ni tako. Res je struktura v osnovni obliki lepša in razumljivejša, vendar problemi nastanejo, ko je treba podatke pridobiti iz tega dokumenta.

V relacijski bazi pridemo do enega zapisa zelo hitro. Vzemimo za primer en artikel iz računa. V bazi NoSQL bi se morali sprehoditi čez seznam vseh artiklov in nato izločiti podatek, ki ga iščemo. Na drugi strani pa nam relacijske baze vrnejo takoj nek specifičen podatek, brez da bi ga morali izločiti.

```
{
    "Naziv trgovine": "Market Rožna dolina",
    "Naslov trgovine": "Rožna dolina c. IX 17, 1000 Ljubljana",
    "Kontaktna številka": "051284325",
    "Datum in čas nakupa": "07.03.2015 19:41",
    "Številka računa": "160/02012889",
    "Status računa": "Potrjeno",
    "Seznam z nazivi artiklov, količino in njihovo ceno":
        [
            {
                "Naziv izdelka": "Vrečka NOS. MERC. PIKA 2013",
                "Količina(x)": 1,
                "Cena(EUR)": 0.10
            },
            {
                "Naziv izdelka": "VODA ODA. NAR. NEG. 0.5L PET",
                "Količina(x)":6,
                "Cena(EUR)":  0.32
            }
        ],
    "Znesek nakupa brez DDV(EUR)":2.02,
    "Vrednost DDV(EUR)": 0.9,
    "Skupni znesek z DDV(EUR)": 2.92,
    "Ime in priimek blagajnika": "Janez Novak"
}
```


Slika 3.8: Primer zapisa enega računa v dokumentu MongoDB

Zaradi omenjenih lastnosti podatkovnih baz NoSQL se danes pri izdelavi spletnih strani, kjer se zahteva celotna vsebina na eni strani, priporoča uporabo omenjenih podatkovnih baz.

3.2.3 Avtentikacija podatkov pri prijavi

Na zalednem podsistemu imamo implementirano funkcijo, ki preverja, ali so podatki pri prijavi na spletni strani pravilni. Proces avtentikacije poteka skozi tri funkcije:

- `login`

- `isInDB`
- `createToken`

Prva funkcija služi zgolj za posredovanje podatkov drugima dvema funkcijama in združevanje podatkov iz teh dveh funkcij. Funkcija `isInDB` na sliki 3.9 nam služi za preverjanje, ali so podatki, ki jih spletna stran posreduje preko klica API `login`, pravi. Kot je razvidno na sliki 3.9, funkcija dobi v parametru objekt, ki vsebuje uporabniško ime in geslo. Nato funkcija poišče v podatkovni bazi uporabnika z uporabniškim imenom, ki ga najde v posredovanem objektu.

Če ima podatkovna baza MongoDB shranjenega uporabnika z iskanim uporabniškim imenom, bo funkcija prejela vse podatke o iskanem uporabniku.

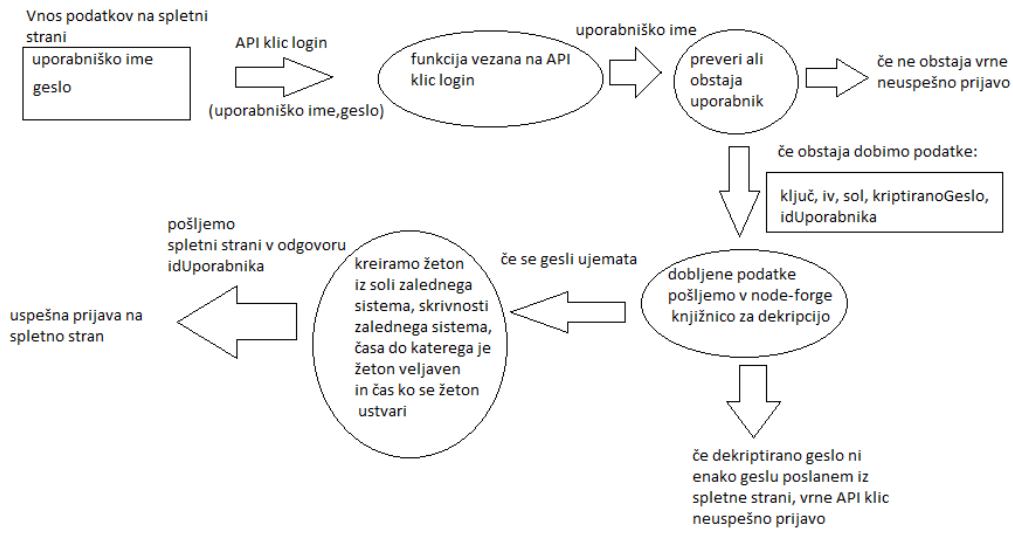
```
function isInDB(userObject){
        var d = deferred();
        var checkInDB = model
        .find({"username":userObject.username})
        .exec();
        var sendObject = {};
        checkInDB.then(
            function(result){
                if(result==''){
                    sendObject.validated = false;
                    d.resolve(sendObject);

                }else{
                    var userData = JSON
                    .parse(JSON.stringify(result[0]));
                    var encryptedData = {};
                    encryptedData.key = userData.key;
                    encryptedData.iv = userData.iv;
                    encryptedData.cipher_text = userData.password;
                    var plainTextPassword = decrypt(encryptedData);
                    if(userObject.password == plainTextPassword.data)
                    {
                        sendObject.validated = true;
                        sendObject.id = userData.id;
                        d.resolve(sendObject);
                    }
                    else
                    {
                        sendObject.validated = false;
                        d.resolve(sendObject);
                    }
                }
            },
            function(error){
                d.reject(error);
            }
        );
        return d.promise;
    }
```

Slika 3.9: Programska koda funkcije `isInDB` v zalednem sistemu

Nato iz podatkov izluščimo:

- kriptirano geslo
- vrednost `iv`, ki je ena izmed dveh vrednosti, s katerima je geslo dekriptirano
- vrednost `key`, ki je ena izmed dveh vrednosti, s katerima je geslo dekriptirano

Izluščene podatke shranimo v objekt. Ta objekt nato pošljemo knjižnici, ki jo uporabljamo za enkripcijo in dekripcijo podatkov, imenovano `node-forge` [9]. Knjižnica dekriptira kriptirano geslo in vrne geslo v golem besedilu. Če se geslo ujema z geslom, ki ga funkcija `isInDB` izlušči iz objekta v parametru, bo funkcija `isInDB` sporočila funkciji `login`, da so podatki pravi. Zraven pošlje še identifikacijsko številko uporabnika, ki jo nato pošlje funkcija `login` spletni strani. Opisani postopek je mogoče videti na sliki 3.9. Poleg prikaza na sliki 3.9 je vse navedeno ponazorjeno s skico na sliki 3.10.



Slika 3.10: Skica ponazoritve izvajanja postopka s slike 3.9

V primeru, da so podatki uporabnika pravi, se pred dokončno avtentikacijo ustvari unikaten `žeton`. Žeton se ustvari iz sledečih podatkov:

- trenutni čas v milisekundah
- čas v milisekundah, do katerega je žeton veljaven
- identifikacijske številke uporabnika
- skrivnost je pojem, ki v računalništvu predstavlja naključen niz znakov, za katere ve le zaledni podsistem
- sol je pojem, ki enako kot skrivnost v računalništvu predstavlja naključen niz znakov, za katere ve le zaledni podsitem, a s to razliko, da je ta niz soli daljši od niza skrivnosti.

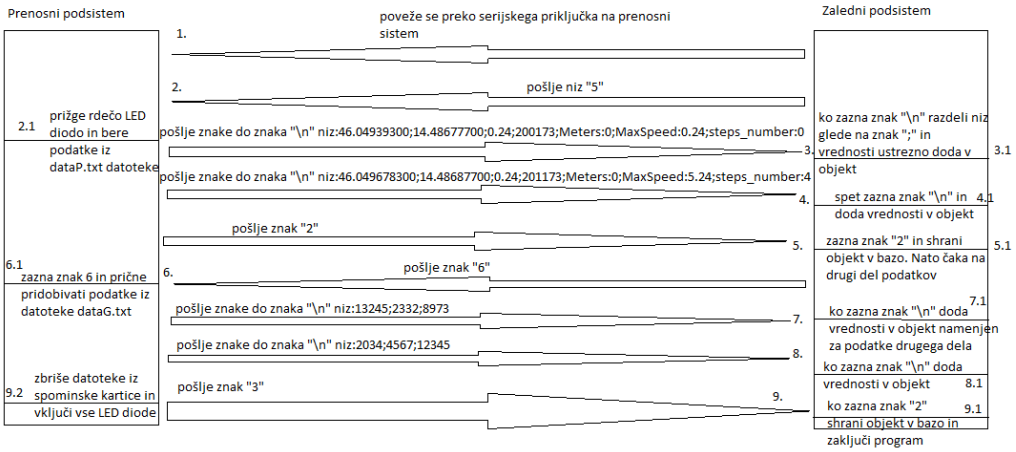
Vse te podatke shranimo v objekt, ki ga pretvorimo v en niz. Niz nato kriptiramo s knjižnico `node-forge` s soljo in skrivnostjo, ki ju ve le zaledni podsistem. Žeton, ki se ustvari iz navedenih podatkov, pošlje zaledni podsistem nazaj spletni strani. Vsakič ko uporabnik želi novo vsebino, se njegov žeton dekriptira in preveri, ali je njegov žeton še veljaven. Programska koda za generiranje žetona je prikazana na sliki 3.11.

```
function createToken (apiSecret,apiSalt,userID){
                var data = {};
                var cas = new Date();
                cas.setTime(cas.getTime()+(1000*60*60));//milisekunde
                data.expiration =cas.getTime();
                data.secret = apiSecret;
                data.salt = apiSalt;
                data.id = userID;
                var passPhrase = (JSON.stringify(data));
                var encryptedObj = encrypt(passPhrase,apiSecret,apiSalt);
                return encryptedObj;


        }
```

Slika 3.11: Programska koda funkcije `createToken` za ustvarjanje žetona

3.2.4 Prenos podatkov med prenosnim podsistemom in zalednim podsistemom

V tem podpoglavju je predstavljen postopek prenosa podatkov med podsistemoma, ki je v osnovnih potezah ponazorjen na sliki 3.12. V nadaljevanju je podrobneje opisan tekstovno in ponazorjen s slikami.



Slika 3.12: Skica ponazoritve postopka prenosa podatkov med prenosnim in zalednim podsistemom

Za avtomatiziranje prenosa podatkov iz prenosnega podsistema na zaledni podsistem je bilo treba spremeniti nastavitve Raspberry Pi. Najprej je bilo treba preurediti določeno sistemsko datoteko. Tako smo na Raspberry Pi v direktorij `/etc/udev/rules.d` dodali datoteko, imenovano `10-Arduino.rules`, v katero smo zapisali to, kar je razvidno iz slike 3.13.

```
1 SUBSYSTEM=="usb", ATTRS{serial}=="55338343539351212011", RUN+="/home/pi/diplomska/test.sh", ATTRS{product}=="Arduino Uno", OWNER=="pi"
```

Slika 3.13: Konfiguracija datoteke za avtomatsko detekcijo naprave Arduino Uno na Raspberry Pi 2, model B

Prvi parameter na sliki 3.13 nam pove, na katerem priključku se bo prenosni podsistem priključil. Drugi parameter, imenovan ATTRS, nam pove serijsko številko naprave. Tretji parameter pove, kaj naj se izvede.

V našem primeru je to skripta Linux, ki je locirana na navigacijski poti `/home/pi/diplomska/test.sh`. V tej skripti je enostaven ukaz, ki požene skripto Python.

```
1 sudo python /home/pi/diplomska/mongotest.py
```

Slika 3.14: Ukaz v skripti, ki se zažene ob priklopu prenosnega podsistema na Raspberry Pi 2, model B

Predzadnji parameter nam pove ime izdelka, zadnji parameter pa ime uporabnika, preko katerega naj se skripta izvede. V naslednjih slikah bomo prikazali logiko, po kateri se podatki prenesejo med podsistemoma in kako se vnesejo v podatkovno bazo.

Na sliki 3.15 je programska koda, ki deluje tako, da preveri, na kateri absolutni poti je priklopljena naprava. Za naprave Arduino je značilno, da se priklopijo na poti `/dev/ttyACM*`, kjer znak `'*'` predstavlja katerokoli število. V primeru na sliki 3.15 se bo izvršila koda, če bo zaznana naprava na poti `/dev/ttyACM0`. Če bo ta pot pravilna, se bo zaledni podsistem povezal na serijski port s hitrostjo 9600-bps in takoj poslal prenosnemu podsistemu znak `'5'`. Tukaj smo si izbrali nek poljuben znak, ki nima nobenega bistvenega pomena.

Na drugi strani se na prenosnem podsistemu zažene druga koda, ki preveri, ali obstaja datoteka na spominski kartici, v kateri so hranjeni določeni podatki. Če datoteka obstaja, bo prenosni podsistem čakal na znak s strani zalednega podsistema(v našem primeru znak `'5'`, kar je razvidno iz slike 3.17). Takoj zatem, ko prenosni podsistem prejme znak `'5'`, vklopi rdečo LED-diodo in začne brati podatke iz datoteke `dataP.txt`, ki jo najdemo na spominski kartici. Te podatke(posamezni znaki) pošilja drugega za drugim zalednemu podsistemu. Vsakič ko se posamičen podatek pošlje, prenosni podsistem počaka, da je podatek zagotovo poslan. Ukaz, ki skrbi za to, je `Serial.flush()`. Opisan postopek je viden na sliki 3.17.

```
stringName = "/dev/ttyACM0"
stringName1 = "/dev/ttyACM1"
dateToday = datetime.datetime.utcnow()#datetime.datetime.now().isoformat()
if (os.path.exists(stringName)):
    #se izvede en del kode
    time.sleep(15)
    ser = serial.Serial(stringName,9600,timeout=0)
    ser.flushInput()
    ser.flushOutput()
    time.sleep(2)
    ser.write('5')
    vrstica = ""
    speeds = []
    obi = createEmptyObject(dateToday)
    while 1:
        if (ser.inWaiting()>0):
            znak = str(ser.read())
            if(znak != '\n'):
                vrstica+=znak
            else:
                vrstica = vrstica.strip()
                if(vrstica =="2"):
                    if(obi['average_speed'] == 0):
                        obi['average_speed']=0
                    else:
                        obi['average_speed']= obi['average_speed']/len(speeds)
                    bla = insertIntoDatabase(obi)
                    break
                if(vrstica != "1"):
                    tabelaVrednosti = vrstica.split(";")
                    speeds.append(1)
                    obi = putDataIntoObject(tabelaVrednosti,obi)
                    vrstica=""
                else:
                    vrstica = ""
    obiVib = createVibObject(dateToday)
```

Slika 3.15: Prvi del kode za prenos podatkov med prenosnim podsistemom in zalednim podsistemom

Istočasno kot prenosni podsistem pošilja podatke, na drugi strani zaledni podsistem na te podatke čaka. Ko dobi podatke, jih združuje v posamezen niz znakov, dokler ne prejme znaka `'/n'`. Takrat preveri, ali je zapis v vrstici enak nizu `'1'` ali nizu `'2'`. Če ni enak nobenemu pogoju, se podatki v vrstici dodajo objektu `obi`, ki hrani vse podatke o vadbi. V primeru, da je niz enak

```
obiVib = createVibObject(dateToday)
ser.write('6')
can = False
vrstica=""
while 1:
    if(ser.inWaiting()>0):
        znak = str(ser.read())
        if(znak != '\n'):
            vrstica+=znak
        else:
            vrstica = vrstica.strip()
            if(vrstica == "4"):
                asdf = insertVibIntoDatabase(obiVib)
                ser.close()
                break
            if(vrstica != "3"):
                se=vrstica.split(";")
                putVibIntoOb(se,obiVib)
                vrstica =""
            else:
                vrstica=""
```

Slika 3.16: Drugi del kode za prenos podatkov med prenosnim podsistemom in zalednim podsistemom

nizu `'2'`, pomeni, da je prvi del podatkov poslan. Od tega trenutka naprej prenosni podsistem čaka na znak `'6'`, ki pomeni, da je zaledni podsistem pripravljen na nov del podatkov. Prvi del podatkov se navezuje na datoteko `dataP.txt`. V omenjeni datoteki imamo strukturo podatkov, kot je razvidna na sliki 3.18. Podatki v vrsticah so ločeni z znakom `';'` in vsaka vrstica ima sledečo strukturo:

- koordinate zemljepisne širine
- koordinate zemljepisne višine
- trenutna hitrost (v km/h)
- čas vadbe (v milisekundah)
- razdalja med trenutno in prejšnjo točko (v metrih)
- največja hitrost, dosežena do trenutka zapisa (v km/h)

- število korakov

V strukturi podatkov imamo tudi podatek o številu korakov, kljub temu, da

```
if(Serial.read() == '5'){
digitalWrite(LEDred,HIGH);
if(SD.exists("dataP.txt"))
{

  myFileP= SD.open("dataP.txt");
  if(myFileP){
    while(myFileP.available()){
          Serial.print(char(myFileP.read()));
          Serial.flush();
          }
          Serial.println("2");
          Serial.flush();
          myFileP.close();
          while(true){
            if(Serial.read() == '6'){
              break;
            }
          }
          myFileG = SD.open("dataG.txt");
          while(myFileG.available()){
            Serial.print(char(myFileG.read()));
            Serial.flush();
          }
          Serial.println("4");
          Serial.flush();
          digitalWrite(LEDred,LOW);
          digitalWrite(LEDblue,HIGH);
          myFileG.close();
          SD.remove("dataP.txt");
          SD.remove("dataG.txt");
          digitalWrite(LEDSTART,HIGH);
          digitalWrite(LEDred,HIGH);
      }
   }
  }
}
```

Slika 3.17: Del programske kode za napravo Arduino za pošiljanje podatkov

```
46.04939300;14.48677700;0.24;200173;Meters:0.00;MaxSpeed:0.24;steps_number:0
46.04939300;14.48677100;0.59;202171;Meters:0.52;MaxSpeed:0.59;steps_number:0
46.04939300;14.48677400;0.28;204167;Meters:0.29;MaxSpeed:0.28;steps_number:0
46.04939300;14.48678100;0.59;206164;Meters:0.52;MaxSpeed:0.59;steps_number:0
46.04939300;14.48678300;0.17;208178;Meters:0.15;MaxSpeed:0.17;steps_number:0
46.04939300;14.48678600;0.89;210165;Meters:0.22;MaxSpeed:0.89;steps_number:0
46.04939300;14.48679400;0.46;212169;Meters:0.66;MaxSpeed:0.46;steps_number:0
46.04938900;14.48679800;0.87;214166;Meters:0.48;MaxSpeed:0.87;steps_number:0
46.04938900;14.48679900;0.48;216169;Meters:0.07;MaxSpeed:0.48;steps_number:0
46.04938500;14.48679700;0.93;218165;Meters:0.24;MaxSpeed:0.93;steps_number:0
46.04938500;14.48680100;0.46;220164;Meters:0.29;MaxSpeed:0.46;steps_number:0
46.04938500;14.48680300;0.78;222163;Meters:0.15;MaxSpeed:0.78;steps_number:0
46.04938500;14.48680600;0.70;224164;Meters:0.22;MaxSpeed:0.70;steps_number:0
46.04938100;14.48680800;0.81;226169;Meters:0.59;MaxSpeed:0.81;steps_number:0
46.04937700;14.48681400;0.87;228168;Meters:0.58;MaxSpeed:0.87;steps_number:0
46.04937400;14.48681800;0.39;230163;Meters:0.68;MaxSpeed:0.39;steps_number:0
46.04937400;14.48682100;0.80;232168;Meters:0.22;MaxSpeed:0.80;steps_number:0
46.04937000;14.48682500;0.30;234167;Meters:0.29;MaxSpeed:0.30;steps_number:0
46.04936200;14.48683500;0.91;236164;Meters:1.52;MaxSpeed:0.91;steps_number:0
46.04935800;14.48683900;0.28;238172;Meters:0.41;MaxSpeed:0.28;steps_number:0
```

Slika 3.18: Podatki v datoteki `dataP.txt`

funkcionalnost ni razvita do popolnosti. Dodali smo jo v strukturo, za lažji nadaljnji razvoj. Program podatke v vrstici loči v tabelo na podlagi znaka `';'`. Te podatke mora nato dodati v objekt, kot je razvidno na sliki 3.19. Ko

```
def putDataIntoObject(tabela,obi):
    mspeed = tabela[5].split(":")
    meters = tabela[4].split(":")
    obi['locations'].append(tabela[0]+","+tabela[1])
    obi['meters_number']+=float(meters[1])
    obi['practice_duration']=int(tabela[3])
    obi['max_speed']=float(mspeed[1])
    obi['average_speed']+=float(tabela[2])
    return obi
```

Slika 3.19: Koda za dodajanje podatkov v objekt

se prenese prvi del, se podatki v objektu `obi` zapišejo v podatkovno bazo na zaledni podsistem. Program se poveže na lokalno instanco podatkovne baze MongoDB. Podatke zapiše v podatkovno bazo `stalker` v zbirko `practice`. Ta postopek je viden na sliki 3.20.

Po zapisu podatkov v podatkovno bazo zaledni podsistem pošlje znak `'6'`, s katerim sporoča, da je pripravljen na podatke o vibracijah. Postopek je enak kot pri pošiljanju prvega dela podatkov, razlikujejo se le podatki, ki so preneseni. V drugem delu je struktura podatkov sledeča:

```
def insertIntoDatabase(objectToSave):

    database = connectToMongo()
    db = database.stalker
    tat = database.practice.insert(objectToSave)
    return
def connectToMongo(): #vrne instanco baze
    location = "localhost"
    port=27017
    database = "stalker"
    table = "practice"
    client = MongoClient(location, port)
    db = client.stalker # this is database
    practiceTable = db['practice']
    return db
```

Slika 3.20: Funkcija za dodajanje vrednosti v objekt(insertIntoDatabase) in funkcija za dodajanje objekta v podatkovno bazo MongoDB (connectToMongo)

- sprememba pospeška po osi x
- sprememba pospeška po osi y
- sprememba pospeška po osi z

Podatki o vibracijah se nahajajo na spominski kartici prenosnega podsistema, natančneje v datoteki `dataG.txt`. Podatki se spet združujejo v vrstice in razdelijo v tabelo z znakom `';'`. Tabelo se na enak način posreduje funkciji, ki podatke shrani v objekt. To funkcijo prikazuje slika 3.22. Podatke se shranjuje v podatkovno bazo s skoraj enako funkcijo kot na sliki 3.20, a z razliko, da se namesto v zbirko `practice` shranjuje v zbirko `vibrations`.

Zapis v zbirko `vibrations` se zgodi, ko prenosni podsistem pošlje niz, ki vsebuje znak `'4'`. Takoj ko je niz poslan, izključi rdečo LED-diodo in odstrani datoteki `dataP.txt` in `dataG.txt` iz spominske kartice. Ko sta datoteki odstranjeni, se vključijo vse LED-diode. S tem prenosni podsistem naznani, da je celoten prenos podatkov končan (slika 3.17).

```
-1205;4200;-12341
-1212;4315;-12271
-1217;4313;-12354
-1208;4303;-12283
-1220;4301;-12347
-1217;4328;-12284
-1227;4315;-12275
-1202;4277;-12335
-1191;4299;-12308
-1185;4325;-12282
-1251;4326;-12292
-1207;4316;-12267
-1220;4326;-12278
-1204;4302;-12329
-1176;4313;-12307
-1175;4317;-12293
-1223;4312;-12334
-1190;4298;-12361
-1242;4340;-12300
-1189;4309;-12273
-1185;4314;-12293
```

Slika 3.21: Primer podatkov v datoteki `dataG.txt`

```
def putVibIntoOb(tabe,obiS):
    obiS['x_axis'].append(tabe[0])
    obiS['y_axis'].append(tabe[1])
    obiS['z_axis'].append(tabe[2])
    return obiS
```

Slika 3.22: Funkcija, s katero dodajamo podatke o vibracijah v objekt

Poglavje 4

Prenosni podsistem za beleženje aktivnosti s tipali

Vloga prenosnega podsistema je beleženje aktivnosti med rekreacijo. Podatke o aktivnosti beležimo s pomočjo naprave Arduino Uno R3 in tipali, ki smo jih povezali na napravo Arduino Uno R3. Podatke o lokaciji, hitrosti, razdalji opravljeni med rekreaciji pridobiva modul GPS. Podatke o vibracijah pa zajamemo z pospeškometrom. Vse skupaj pa zapisujemo na spominsko kartico, ki je povezana na napravo Arduino preko modula za spominsko kartico.

4.1 Implementacija prenosnega podsistema

Jedro prenosnega podsistema je naprava Arduino Uno R3. Arduino Uno je mikrokrmilnik, na katerega lahko priklopimo tipala za zajem podatkov iz okolice. Mikrokrmilnik skrbi za ukazovanje tipalom, kaj naj ta zajamejo iz okolice. Njegovo jedro predstavlja čip Atmega328P. Ima 14 priključkov, ki lahko predstavljajo digitalne vhode ali izhode, 6 analognih vhodov, 14 MHz kvarčni kristal, priključek USB, priključek za napajanje, glavo ICSP in tipko reset. Arduino ploščica ima na voljo 23 KB pomnilnika Flash, 2 KB pomnilnika SRAM in 1 KB EEPROM. Pomnilnik flash je uporaben le za shranjevanje programske kode in ostanek je mogoče uporabiti za podatke, ki

jih ustvari Arduino med delovanjem. Vsebuje vse, kar potrebuje mikrokrmilnik za delovanje. Napajamo ga lahko preko:

- priključka USB, ki ga priključimo v računalnik
- priključka za napajanje z adapterjem AC-DC
- priključka za napajanje z baterijo

Slika 4.1: Arduino Uno R3 [17]

"Uno" pomeni v italijanščini "ena". To ime se je izbralo, ker je označevalo pojavitev programskega okolja za ploščice Arduino imenovano "Arduino Software (IDE) 1.0". Arduino Uno je prva ploščica v družini mikrokrmilnikov Arduino. V letih se je razvilo mnogo drugih ploščic.

Na sliki 4.2 je seznam vseh ploščic z njihovimi specifikacijami. Dobra stran raznolikosti ploščic je v tem, da z enako kodo lahko uporabimo skoraj katerokoli ploščico. Razlika je v tem, ali imamo opraviti z mikrokrmilnikom, ki temelji na arhitekturi ARM. Npr. Arduino Due uporablja mikrokrmilnik,

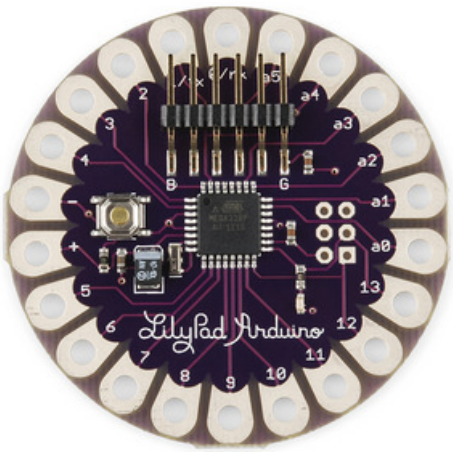
Name	Processor	Operating/Input Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [KB]	SRAM [KB]	Flash [KB]	USB	UART
ArduinoBT	ATmega328P	5 V / 2.5-12 V	16 MHz	6/0	14/6	1	2	32	-	1
Due	ATSAM3X8E	3.3 V / 7-12 V	84 MHz	12/2	54/12	-	96	512	2 Micro	4
Esplora	ATmega32U4	5 V / 7-12 V	16 MHz	-	-	1	2.5	32	Micro	-
Ethernet	ATmega328P	5 V / 7-12 V	16 MHz	6/0	14/4	1	2	32	Regular	-
Fio	ATmega328P	3.3 V / 3.7-7 V	8 MHz	8/0	14/6	1	2	32	Mini	1
Gemma	ATtiny85	3.3 V / 4-16 V	8 MHz	1/0	3/2	0.5	0.5	8	Micro	0
Leonardo	ATmega32U4	5 V / 7-12 V	16 MHz	12/0	20/7	1	2.5	32	Micro	1
LilyPad	ATmega168V ATmega328P	2.7-5.5 V / 2.7-5.5 V	8MHz	6/0	14/6	0.512	1	16	-	-
LilyPad SimpleSnap	ATmega328P	2.7-5.5 V / 2.7-5.5 V	8 MHz	4/0	9/4	1	2	32	-	-
LilyPad USB	ATmega32U4	3.3 V / 3.8-5 V	8 MHz	4/0	9/4	1	2.5	32	Micro	-
Mega 2560	ATmega2560	5 V / 7-12 V	16 MHz	16/0	54/15	4	8	256	Regular	4
Mega ADK	ATmega2560	5 V / 7-12 V	16 MHz	16/0	54/15	4	8	256	Regular	4
Micro	ATmega32U4	5 V / 7-12 V	16 MHz	12/0	20/7	1	2.5	32	Micro	1
Mini	ATmega328P	5 V / 7-9 V	16 MHz	8/0	14/6	1	2	32	-	-
Nano	ATmega168 ATmega328P	5 V / 7-9 V	16 MHz	8/0	14/6	0.512 1	1 2	16 32	Mini	1
Pro	ATmega168 ATmega328P	3.3 V / 3.35-12 V 5 V / 5-12 V	8 MHz 16 MHz	6/0	14/6	0.512 1	1 2	16 32	-	1
Pro Mini	ATmega328P	3.3 V / 3.35-12 V 5 V / 5-12 V	8 MHz 16 MHz	6/0	14/6	0.512	1	16	-	1
Uno	ATmega328P	5 V / 7-12 V	16 MHz	6/0	14/6	1	2	32	Regular	1
Yún	ATmega32U4 AR9331 Linux	5 V	16 MHz 400MHz	12/0	20/7	1	2.5 16MB	32 64MB	Micro	1
Zero	ATSAMD21G18	3.3 V / 7-12 V	48 MHz	6/1	14/10	-	32	256	2 Micro	2

Slika 4.2: Primerjava modelov Arduino med seboj [18]

ki je zgrajen na arhitekturi ARM, zato je treba imeti drugačne knjižnice za tipala.

Dobra plat raznolikosti ploščic je v tem, da je vsaka ploščica namenjena za specifičen projekt. Arduino Uno je namenjen za majhne in prenosne projekte, medtem ko je Arduino Mega 2560 primernejši za večje projekte, saj ima več priključkov in tudi več spomina za programsko kodo. Za zares majhne

projekte imamo možnost uporabiti LilyPad Arduino, ki si ga lahko prišijemo na majico, ali pa Arduino Nano, s katerim lahko naredimo svoje vezje. Specifikacije, na katere smo najbolj pozorni, so hitrost mikrokrmilnika, velikost pomnilnika Flash in število priključkov za komponente. Pri izbiri ploščice je treba biti posebej pozoren, saj nepravilna odločitev lahko povzroči veliko nevšečnosti. Če si npr. izberemo projekt, za katerega je značilno, da bo mobilen, se zelo hitro zgodi, da ne ostane več prostih priključkov. Drugič spet je priključkov dovolj, a so programska koda in knjižnice, ki jih potrebujemo za delovanje tipal preprosto preobsežne. Pomnilnik, ki je zadolžen za hranjenje kode, preprosto ne premore toliko prostora. Ker smo potrebovali ploščico, ki je majhna in takoj po nakupu pripravljena na priključitev tipal, smo se odločili za uporabo ploščice Arduino Uno R3.



Slika 4.3: LilyPad Arduino [19]

Arduino ploščice so zelo priljubljene pri projektih `naredi sam`. Na spletu je mogoče dobiti ogromno primerov, iz katerih lahko izluščimo koristno kodo za svoje projekte. Zaradi svoje priljubljenosti je tudi veliko knjižnic za module, napisanih prav za ploščice Arduino.

Slika 4.4: Arduino Nano [20]

4.2 Tipala za zajem podatkov iz okolice

Jedro prenosnega podsistema je naprava Arduino, a podatke iz okolice pa pridobivajo tipala. Da nam tipala posredujejo podatke okolice moramo vzpostaviti pravilno komunikacijo med napravo Arduino in vsemi tipali priključenimi na napravo.

4.2.1 Knjižnice

Za komunikacijo s tipali smo uporabljali knjižnice. To je programska koda, ki je že napisana in jo je napisala tretja oseba ali pa je bila vključena v okolje za pisanje kode Arduino IDE. Napisane so v programskih jezikih C++ ali C. Njihova funkcija je sporazumevanje s tipali. Ker je za pisanje takih knjižnic potrebnega veliko znanja z elektrotehničnega področja, se pogosto uporablja knjižnice tretjih oseb. Po eni strani nočemo "odkrivati tople vode", če že obstaja rešitev, ki nam olajša delo.

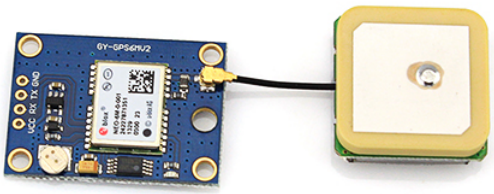
Za diplomsko delo so bile tako uporabljene naslednje knjižnice:

- `TinyGPS`, ki nam pomaga pri komunikaciji z modulom GPS [11]
- `SD`, ki nam omogoča pisanje v in branje iz spominske kartice

- `SoftwareSerial` za komunikacijo s serijskim priključkom
- `SPI`, s katero lahko povežemo mikrokmilnik z drugim mikrokrmilnikom, ali več perifernimi napravami
- `Wire`, ki nam omogoča ukazovanje perifernim napravam, kdaj poslati podatke
- `SparkFun-LSM9DS0 Breakout` za pospeškometer [12]

4.2.2 Globalni sistem GPS za določanje položaja

Globalni sistem za določanje položaja je satelitski navigacijski sistem, ki ponuja podatke o lokaciji in času v kakšnih koli vremenskih pogojih. Podatke dobimo, če imamo nemoten signal do vsaj štirih satelitov. Sistem so razvili Američani, ki ga tudi vzdržujejo in omogočajo dostop vsakomur, ki ima sprejemnik GPS. Projekt se je začel leta 1973, ko so se hoteli rešiti omejitev, ki so jih predstavljali obstoječi navigacijski sistemi. Razvilo ga je ministrstvo za obrambo Združenih držav Amerike. Projekt je prvotno uporabljal 24 satelitov. Leta 2000 je kongres Združenih držav Amerike sprejel pobudo za modernizacijo.



Slika 4.5: Modul GPS GY-NEO6MV2 [21]

Sistem deluje po načelu časa. Sateliti imajo stabilne atomske ure, ki so sinhronizirane med seboj in z urami na Zemlji. Vsako odstopanje na Zemlji se popravlja vsakodnevno, prav tako pa so ure satelitov natančno nadzorovane. Tudi sprejemniki GPS imajo ure, vendar te niso sinhronizirane z urami na Zemlji oz. sateliti, zato niso najnatančnejše. Sateliti neprekinjeno pošiljajo svoj čas in lokacijo. Sprejemnik preverja različne satelite in preko določenih enačb določi natančno lokacijo sprejemnika ter njegovo odstopanje od pravega časa. Za delovanje potrebuje sprejemnik minimalno štiri satelite, da lahko izračuna štiri neznanke, tri koordinate in odstopanje ure od časa satelitov [13].

Moduli GPS (eden od njih je uporabljen v prenosnem podsistemu) večinoma potrebujejo več časa za vzpostavitev komunikacije s sateliti kot pametni telefoni. Pametni telefoni namreč lahko poleg GPS uporablja tudi mobilne podatke. Ko slednje vključimo, se lahko naprava poveže na strežnike A-GPS, kjer se hranijo orbitalne informacije. Pogosto je na lokacijah signal med sprejemnikom in sateliti slabši. To se dogaja v mestih zaradi velikih zgradb, od katerih se signal odbija, ali pa v gozdu, kjer signal motijo drevesa. V takšnih primerih se naprave, ki omogočajo sistem A-GPS, povežejo na te strežnike in dobijo takoj informacije o času in poziciji satelitov. Zato tudi naši pametni telefoni pridobijo GPS signal v roku 3-5 sekund, medtem ko so na drugi strani moduli GPS (uporabljen v diplomskem delu), ki v povprečju potrebujejo 25-50 sekund, da pridobijo povezavo.

Modul GPS je na prenosnem podsistemu priključen kot digitalni vhod na nožici 2 in kot digitalni izhod na nožici 3. Kot je to razvidno na sliki 4.7. Ti dve nožici sta pravzaprav priključka serijskega vmesnika, zato je komunikacija teče po serijski povezavi. Priključek `Rx` skrbi za prejemanje podatkov, priključek `Tx` pa za pošiljanje podatkov.

Na sliki 4.6 je mogoče videti programsko kodo, s katero komuniciramo z modulom GPS. Poudaril je treba, da so vse spremenljivke, ki niso inicializirane in deklarirane v prikazani kodi, deklarirane in inicializirane ob zagonu prenosnega podsistema. V programski kodi na sliki 4.6 je mogoče videti

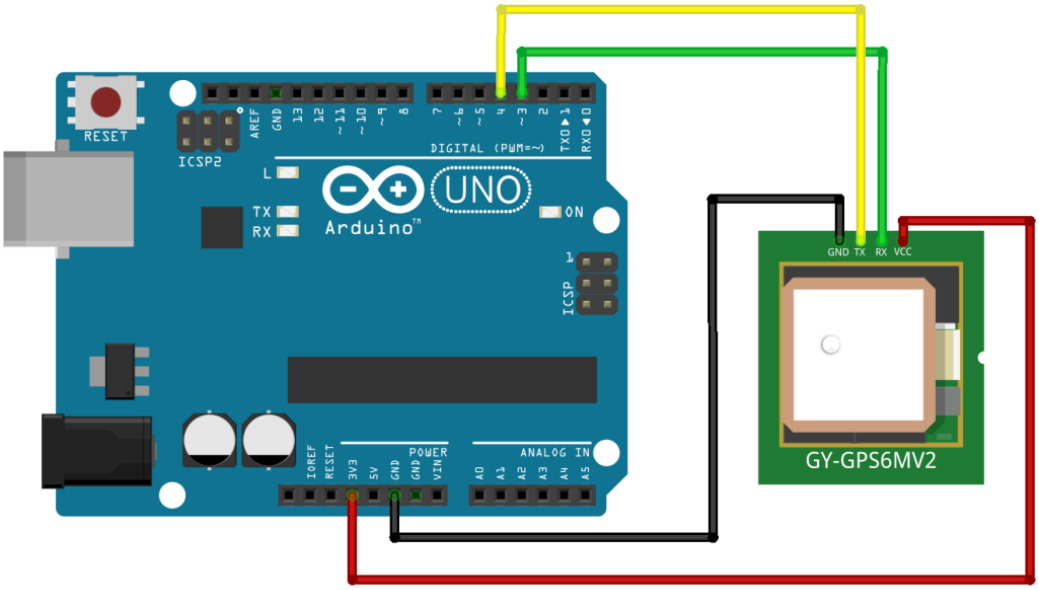
dve pomembni spremenljivki za komunikacijo z modulom GPS. To sta spremenljivki `nss` in `gps`. Prva predstavlja objekt knjižnice `SoftwareSerial`, ki se poveže z modulom GPS. V našem primeru je povezana s hitrostjo 9600-

```
while (nss.available() && startPractice && !over)
{
  digitalWrite(LEDred,LOW);
  digitalWrite(LEDblue, HIGH);
  buttonstate_ = analogRead(buttonstate2);
  if(buttonstate_ > 600){
    digitalWrite(LEDblue,LOW);
    startPractice = false;
    digitalWrite(LEDSTART,LOW);
    over = true;
    myFileP.close();
    myFileG.close();
    break;
  }
  refference = millis();
  c = nss.read();
  if (gps.encode(c) && abs(refference-time_practice)>=1500)
  {
    myFileP = SD.open("dataP.txt", FILE_WRITE);
    myFileG = SD.open("dataG.txt", FILE_WRITE);
    time_practice = refference;
    gps.f_get_position(&flat, &flon, &age);
    float currentSpeed = gps.f_speed_kmph();
    if(currentSpeed > maxSpeed){
      maxSpeed = currentSpeed;
    }
    printAccel();
    if(myFileP){
      String vrstica = "";
      vrstica=String(flat,8);
      vrstica+=";"+String(flon,8);
      vrstica+=";"+String(currentSpeed,2);
      vrstica+=";"+String(refference);
      if(prevLat && prevLon){
        vrstica+=";Meters:"+String(gps.distance_between(flat,flon,prevLat,prevLon));
        prevLat = flat;
        prevLon = flon;
      }else{

        prevLat = flat;
        prevLon = flon;
        vrstica+=";Meters:"+String(gps.distance_between(flat,flon,prevLat,prevLon));
      }
      vrstica+=";MaxSpeed:"+String(maxSpeed);
      vrstica+=";steps_number:"+String(countThresholdOverhaul);
      myFileP.println(vrstica);
      myFileP.close();
```

Slika 4.6: Koda prenosnega podsistema za komunikacijo z modulom GPS

bps, da dobiva podatke. Druga spremenljivka predstavlja objekt knjižnice `TinyGPS`, katere naloga je pretvarjanje pridobljenih podatkov iz spremenljivke `nss` v podatke, primerne za nas (npr. pretvorba podatkov v koordinate lokacije, trenutno hitrost, razdaljo med dvema točkama, itd.). Če v programski kodi na sliki 4.6 prezremo ostale spremenljivke, ki se ne tičejo teh dveh objektov, lahko lažje razumemo komunikacijo z modulom GPS. Najprej preverimo, ali ima GPS modul kaj podatkov, ki jih hrani objekt `nss`. Če so podatki na voljo, se podatke prebere iz objekta `nss`. Prebrane podatke nato kodiramo z objektom `gps` in če so podatki pravilni, bodo kodirani. Ko se podatki kodirajo, se v objektu `gps` shranijo vsi podatki, ki jih je prejel modul GPS. Te podatke nato pridobivamo preko določenih funkcij. Npr. za pridobivanje trenutne lokacije uporabimo funkcijo `get_position` objekta `gps`. V omenjeno funkcijo vključimo tri spremenljivke. Prvi dve sta kazalca na decimalni vrednosti, ki predstavljata koordinato zemljepisne širine in koordinato zemljepisne dolžine. Tretja je kazalec na število, ki predstavlja starost podatka. Ker smo podali kazalce na podatke, lahko knjižnica TinyGPS dodeli vrednosti omenjenim spremenljivkam, ki bodo takoj dostopne v našem programu.

Na sliki 4.7 vidimo primer vezalne sheme med modulom GPS in ploščico Arduino. Ker nam je na napravi Arduino zmanjkovalo serijskih priključkov, smo rumeno žičko, ki gre iz priključka `TX` modula GPS, povezali na priključek `2` naprave Arduino.

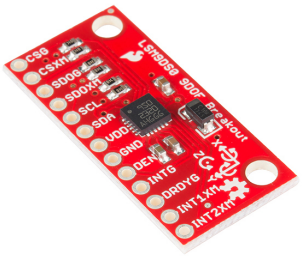


Slika 4.7: Vezalna shema modula GPS z napravo Arduino [26]

4.2.3 Pospeškometer SparkFun-LSM9DS0

Pospeškometer je naprava, ki meri pospešek (gravitacijsko silo). Pravi pospešek ni koordinatni pospešek, saj pravi pospešek meri velikost spremembe hitrosti. Pospeškometer bo v mirnem stanju na ravni podlagi izmeril pospešek 9.81 m/s^2, saj je to gravitacijska sila Zemlje. Ko pa pospeškometri prosto padajo, bodo izmerili vrednost nič zaradi gravitacije zemlje.

Pospeškometri se uporabljajo v različne namene v industriji in znanosti. Zelo občutljivi pospeškometri so sestavni del znotraj navigacijskih sistemov v letalih in raketah. Veliko se jih uporablja za zaznavanje in nadzor vibracij v vrtljivih strojih. Uporabljajo jih tudi v brezpilotnih napravah za stabilizacijo letenja. Danes se največkrat srečamo z njimi v pametnih telefonih in prenosnih tablicah, kjer se jih uporablja za zaznavanje orientacije naprav. Eno osni ali več osni modeli pospeškometrov zmorejo zaznati moč in smer gravitacijske sile kot kvantitetni vektor. S tem se lahko ugotavlja orientacijo, koordinate pospeška, vibracije, udarec itd. Poleg prenosnih naprav ga uporabljajo kontrolerji za igričarske konzole, kjer z zaznavanjem sprememb



Slika 4.8: Pospeškometer, barometer in giroskop SparkFun-LSM9DS0 [22]

pozicije kontrolerja prenese te podatke v igro in temu primerno premakne npr. letalo v igri.

Pospeškometer SparkFun-LSM9DS0, ki je uporabljen v diplomskem delu, ima nožico `INT1XM` povezano na nožico 8 na napravi Arduino. Podatek `INT1XM` nam pove, kdaj so podatki pospeškometra na voljo. Nožici `SCL` in `SDA` imata preko logičnega pretvornika povezani na `SCL` in `SDA` nožici naprave Arduino. Pospeškometer deluje na napetosti med 2,4 in 3,6 V. Podpira tri pospeševalne kanale, tri naklonske kanale in tri kanale za magnetno polje. Na njem lahko merimo gravitacijske sile od 2 G do 16 G, od 2 do 12 Gaussove magnetne lestvice in od 245 do 2000 dps (stopinj na sekundo) naklonske lestvice. Ima 16-bitni izhod za podatke in dva serijska vmesnika (SPI in I2C). V diplomskem delu smo za komunikacijo s pospeškometrom uporabili protokol I2C. Ta nam omogoča, da imamo več različnih podrejenih naprav povezanih na eno nadrejeno napravo. Komunikacija poteka preko sporočil, ki so razdeljene na dva glavna okvirja. Prvi pove na katero napravo naslavlja sporočilo, drugi pa je lahko sestavljen iz več pod okvirjev in vsebuje sporočilo, ki je namenjeno naslovljeni napravi. Sporočila se pošiljajo glede na povezavo na časovni signal `SCL`, ki je za vsako napravo drugačen. Ko je časovni signal doseže vrednost, ki čipu naznani stanje “0”, takrat se podatki berejo iz serijske povezave. Ko pa ima časovni signal vrednost, ki jo čip interpretira kot stanje “1”, se podatki zapišejo na serijsko povezavo.

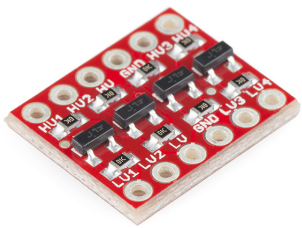
Ker ploščica Arduino običajno deluje na 5 V, pospeškometer pa na 3,3 V, je bilo potrebno dodati digitalni pretvornik napetosti iz 3,3 v 5 V. V nasprotnem primeru bi pospeškometer pošiljal določene podatke, ki bi bili vezani na območje 3,3 V, ploščica Arduino pa bi te podatke lahko interpretirala napačno.

Na sliki 4.10 je programska koda, ki se je uporabljala v prenosnem podsistemu za zajem podatkov iz pospeškometra. Funkcija `calibrate` (na sliki 4.10) nam pomaga, da ob začetku vadbe zajamemo izhodiščne vrednosti osi x,y in z pospeškometra. Te izhodiščne vrednosti se bodo uporabljale za namen zaznavanja korakov. Funkcija najprej preveri ali je pospeškometer pripravljen za zajem podatkov. Če je, pokliče funkcijo, ki zapiše nove vrednosti osi x,y in z v objektu `dof`. Ko nastavi te vrednosti, izračuna še izhodiščno vrednost, ki jo izračuna glede na izraz 4.1.

$$vektorPos = \sqrt[2]{vrednostX^2 + vrednostY^2 + vrednostZ^2} \tag{4.1}$$

Spremenljivka `vektorPos` izraza 4.1 nam poda vektor pospeška. Vrednost `vrednostX` name pove izmerjeno vrednost v točki x. Spremenljivki `vrednostY` in `vrednostZ` nam podata vrednost izmerjeno v točkah y in z.

Vrednosti, ki jih beremo iz pospeškometra so analogne vrednosti pretvorjene v digitalne vrednosti in predstavljajo vrednost sile, ki deluje na pospeškometer. Funkcija `printAccel` (na sliki 4.10) skrbi za zapis vrednosti vibracij v datoteko `dataG.txt`. Najprej funkcija preveri, ali ima pospeškometer



Slika 4.9: Logični pretvornik napetosti iz 3,3 v 5 V in obratno [23]

```
void calibrate()
{
  if (digitalRead(INT1XM))
  {
    // Use the readAccel() function to get new data from the accel.
    // After calling this function, new values will be stored in
    // the ax, ay, and az variables.
    dof.readAccel();
    resultSquare = sqrt(pow(dof.calcAccel(dof.ax),2 )+ pow(dof.calcAccel(dof.ay),2) + pow(dof.calcAccel(dof.az),2));
    calibrated = true;
  }
}


void printAccel()
{

  timeCalc = millis();
  if (digitalRead(INT1XM) && abs(timeCalc - time)>=400)
  {
    time = timeCalc;
    dof.readAccel();
    resultSquareNow = sqrt(pow(dof.calcAccel(dof.ax),2 )+ pow(dof.calcAccel(dof.ay),2) + pow(dof.calcAccel(dof.az),2));
    if(abs(resultSquare-resultSquareNow) > threshold){
      countThresholdOverhaul+=2;
    }
      if(myFileG)
      {
        String vrstica = "";
        vrstica+=String(dof.ax);
        vrstica+=";";
        vrstica+=String(dof.ay);
        vrstica+=";";
        vrstica+=String(dof.az);
        myFileG.println(vrstica);
        myFileG.close();
      }

  }
}
```

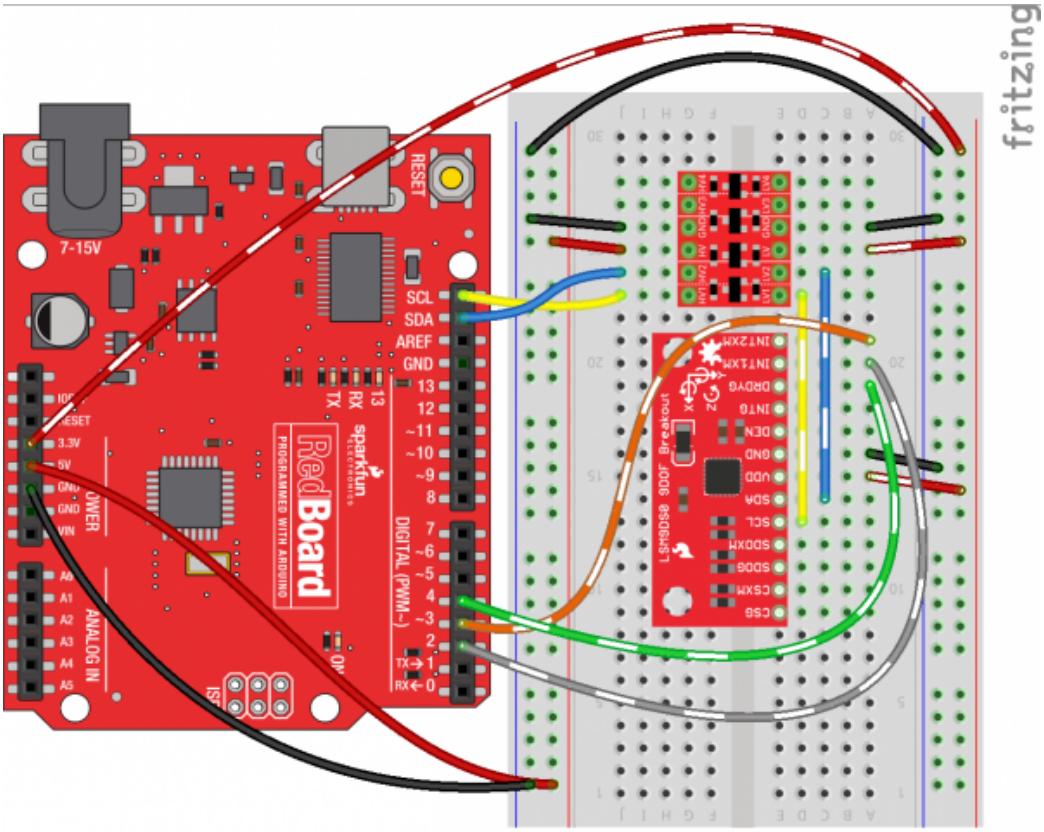
Slika 4.10: Funkcija za nastavitev pospeškometra ob začetku vadbe (calibrate) in funkcija za pridobivanje podatkov iz pospeškometra (printAccel)

podatke na voljo in če je potekel določen čas. Če je pogoj izpolnjen, preveri, ali je spremenljivka `myFileG` inicializirana. Ta spremenljivka je seveda globalna. Če je inicializirana, se podatki zapišejo v datoteko.

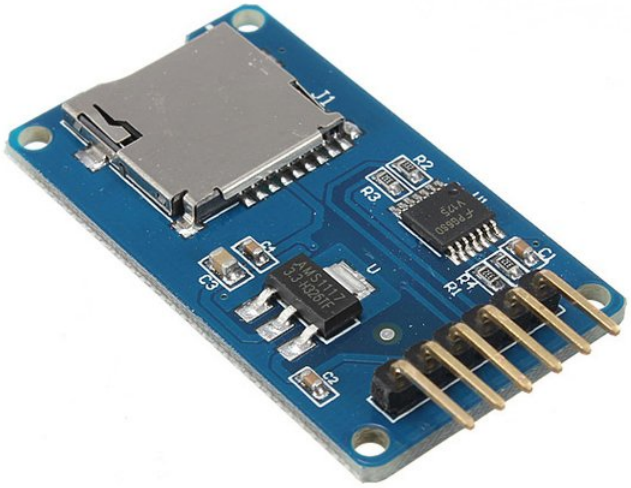
Na sliki 4.11 vidimo primer vezave pospeškometra z napravo Arduino. V diplomskem delu smo sivo, oranžno in zeleno žičko povezali z napravo Arduino na sledeči način:

- sivo žičko smo povezali na priključek `8`
- oranžno žičko smo povezali na priključek `10`
- zeleno žičko smo povezali na priključek `7`

Med razvijanjem sistema smo ugotovili, da podatkov iz priključek `7` in `10` ne bomo potrebovali, saj omenjena priključka pridobivata podatke iz giroskopa in magnetometra. Za morebitni nadaljnji razvoj projekta smo kljub temu pustili omenjena priključka povezana na ploščico Arduino.



Slika 4.11: Primer vezalne sheme pospeškometra z napravo Arduino [25]



Slika 4.12: Adapter Micro card [24]

4.2.4 Modul za spominsko kartico MicroSD

Pri zajemu podatkov iz okolice se je že vnaprej vedelo, da bodo podatki presegali velikost tudi več kot 100 KB. Zaradi tega je bila edina izbira uporaba dodatnega pomnilnika. Na dodatni pomnilnik se je zapisovalo pridobljene podatke iz okolice. V ta namen je uporabljen modul `Micro Card Adapter v1.0` [24] s spominsko kartico `microSD`. Na kartico se shranjujejo vsi podatki o lokaciji, trenutni hitrosti, opravljeni razdalji in času vadbe ter vsi podatki o vibracijah. Modul je na napravi Arduino priključen na nožice 11, 12, 13 in 4. Za komunikacijo in upravljanje spominske kartice uporabljamo knjižnico `SD`. Na sliki 4.13 imamo primer komunikacije s spominsko kartico. V nadaljevanju predstavljamo potek branja iz spominske kartice in zapisovanja v datoteko na spominski kartici.

Modul za spominsko kartico uporablja za komunikacijo z napravo Arduino serijsko vodilo `SPI`. Ta omogoča, da le glavna naprava (v našem primeru

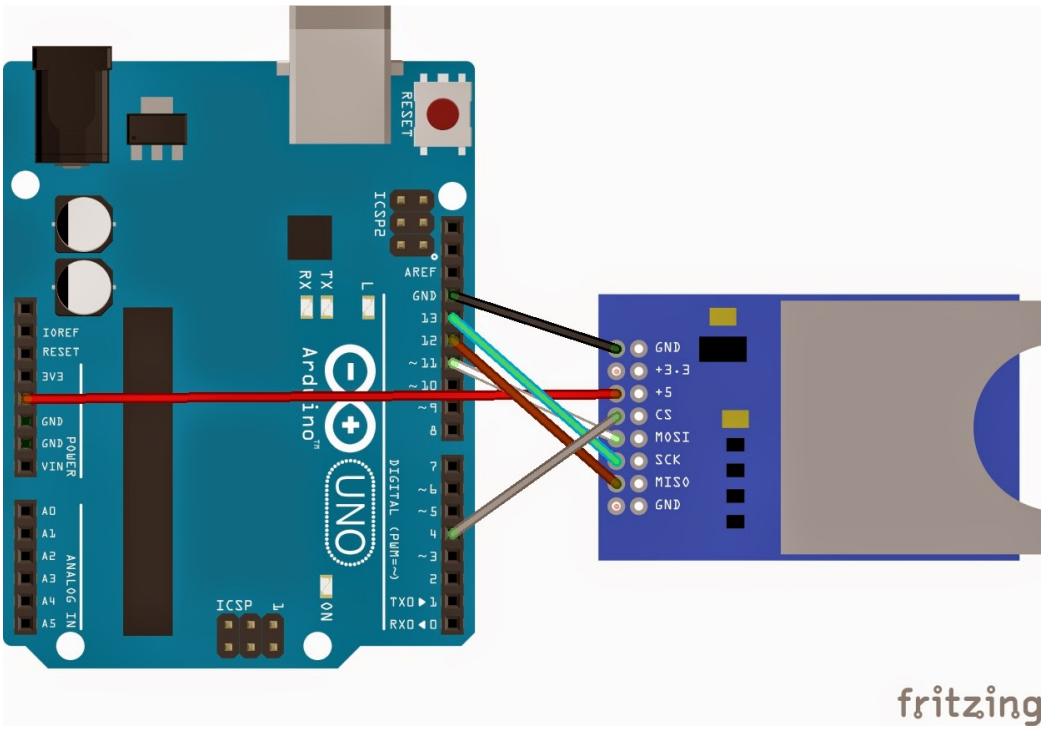
naprava Arduino) začne komunikacijo z ostalimi podrejenimi tipali. Ko želi glavna naprava poslati podatke tipalu, ali prejeti podatke od tipala, mora pravemu `CS` priključku nastaviti vrednost na "0". Ko to stori, nastavi časovni signal na frekvenco, ki je primerna za komunikacijo z izbranim tipalom. Za pošiljanje podatkov uporablja naprava Arduino `MOSI` priključek medtem, ko za branje podatkov iz serijskega vodila uporablja priključek `MISO`.

Najprej zaženemo komunikacijo z modulom preko priključka 4. Nato program preveri, ali obstaja datoteka na spominski kartici. Če obstaja, jo izbriše. Sledita kreiranje in istočasno odpiranje datoteke `test.txt` v načinu za zapisovanje podatkov na spominsko kartico. Postopek, ki smo ga opisali, se zgodi pred zagonom samega programa na napravi Arduino. Ko se program na napravi Arduino zažene, bo preverjal, ali vsebuje globalna spremenljivka `vrstica` manjše število od 10. Dokler bo tako, bo program zapisoval na spominsko kartico "`To je vrstica številka: *`", kjer znak (`'*'`) predstavlja število v globalni spremenljivki `vrstica`. Takoj zatem poveča vrednost spremenljivke `vrstica` za vrednost 1. To ponavlja, dokler spremenljivka `vrstica` ne doseže vrednosti 10. Ko se to zgodi, se datoteka `test.txt` zapre.

```
File testFile;
int vrstica=1;
void setup()
{
  SD.begin(4);
  if(SD.exists("test.txt"){
    SD.remove("test.txt")
  }
  testFile = SD.open("test.txt", FILE_WRITE);
}
void loop()
{
  while(vrstica < 10){
    testFile.println("To je vrstica številka: "+String(vrstica));
    vrstica++;
  }
  testFile.close()
}
```

Slika 4.13: Primer programske kode za komunikacijo s spominsko kartico

Na sliki 4.14 vidimo vezalno shemo med modulom za spominsko kartico in napravo Arduino.



Slika 4.14: Vezalna shema med modulom za spominsko kartico in napravo Arduino [27]

Poglavje 5

Spletni prikaz podatkov

Zajeti podatki se prikazujejo na spletni strani, ki je razvita z ogrodjem AngularJS [2], ki predstavlja vizualni izgled sklada MEAN, ki je uporabljen v sistemu. MEAN predstavlja skupek naslednjih okolij:

- M = MongoDB
- E = ExpressJS
- A = AngularJS
- N = NodeJS

AngularJS je ogrodje, razvito v programskem jeziku JavaScript, ki ga je razvil Google. Omogoča enostaven razvoj spletnih strani, čigar vsebina je vidna znotraj enega spletnega pogleda. Deluje po načelu MVC(Model-Pogled-Kontrola) in je razdeljen v tri dele. Prvi je `model`, ki predstavlja podatke, ki se bodo prikazovali. Drugi je `pogled`, ki skrbi za prikaz vsebine. Zadnja je `kontrola`, v kateri se izvajata vsa logika in procesiranje podatkov.

Vzemimo za primer, da imamo seznam objektov, ki predstavljajo podatke o osebi (ime, priimek, datum rojstva, starost, spol). V primeru načela MVC, bi to predstavljalo model. Torej so to podatki, ki so v ozadju in jih moramo prikazati. Pogled predstavlja spletno stran, ki prikazuje podatke iz modela. To pomeni, da imajo podatki iz modela neposredno povezavo na pogled.

Če na spletni strani prikazane podatke spreminjamo, se bo ta sprememba istočasno zgodila tudi na modelu. Zaradi takega delovanja je preprosteje razvijati strani, pri katerih se nam z interakcijo spremeni le določena vsebina na spletni strani. Tretji del `kontrola` predstavlja del načela MVC, ki povezuje pogled z modelom.

Ogrodje AngularJS uporabljamo znotraj dokumentov HTML, kjer lahko poleg standardnih značk HTML/HTML5 dodamo specifične značke, ki jih prepozna ogrodje. Najpogosteje uporabljene osnovne značke so:

- ng-app
- ng-if
- ng-class
- ng-style
- ng-show/ng-hide
- ng-model
- ng-controller

```
<body ng-app="stalker" ng-controller="AppCtrl" style="background-color:#00a86b;">
    <div class="row">
        <div >
            <img src="img/Sports_Concussion_Graphic_415885_7.png"/ class="headerSize">
        </div>
    </div>
    <div class="row">
        <div class="col">
            <main ng-view="" autoscroll="true" class="" style="position:relative;"></main>
        </div>
    </div>
```

Slika 5.1: Glavna stran, na kateri se prikazuje vsa vsebina spletne strani

Ogrodje AngularJS je razdeljeno na več različnih delov, kjer ima vsak svojo pomembno vlogo:

- kontrola, skrbi za povezovanje podatkov med pogledom in modelom

- storitev, skrbi za logiko in procesiranje podatkov (storitev lahko vključimo v katerikoli kontrolo)
- z direktivo lahko naredimo svoje značke, ki jih dodamo v elemente HTML (direktivo lahko vključimo v katerokoli element `HTML`)

Razdelitev logike v svoj del pomeni, da se v primeru, ko kliknemo na tipko, najprej ta klik zazna v kontroli, vendar zaradi dobre prakse prepustimo delovanje po kliku storitvam.

Na sliki 5.1 imamo del kode `HTML` naše spletne strani. Na njej lahko vidimo značke:

- np-app
- np-controller
- np-view

Z značko `np-app` povemo ogrodju AngularJS, ime naše spletne aplikacije (spletna stran). V znački `ng-controller` navedemo ime kontrole, ki je vezana na trenutni element `HTML`. Značka `ng-view` sporoči ogrodju AngularJS, da se bo znotraj elementa, kjer se nahaja omenjena značka, prikazovala celotna vsebina.

Omeniti je treba možnost hierarhije kontrol. Na sliki 5.1 imamo kontrolo `AppCtrl`, ki bo glavna kontrola za vso vsebino spletne strani. Ta kontrola ima lahko več podrejenih kontrol, ki se navezujejo na specifične poglede. Takšno delovanje je zelo koristno, ker lahko v glavni kontroli `AppCtrl` povežemo podatke različnih podrejenih kontrol.

5.1 Prijavna stran

Pri prijavi imamo dva elementa za vnos podatkov in en sam gumb, kar prikazuje slika 5.2. Za boljšo predstavo je mogoče videti zaslonsko sliko v poglavju 6 na sliki 6.2. Opazimo lahko, da pogled za prijavo na sliki 5.2 vsebuje znački `ng-model` in `ng-submit`.

```
<div class="container-fluid">
    <div class="form-group">
        <div style="text-align:center;">
            <h1 class="text">
                Welcome to Stalker
            </h1>
        </div>

        <div class="loginAlign">
            <div >
                <form ng-submit="vm.loginUser(user)">
                    <input name="username" type="text" class="form-control"
                        ng-model="user.username" placeholder="{{vm.usernameInput}}"/>
                    <br/>
                    <input name="password" type="password" class="form-control"
                        ng-model="user.password" placeholder="{{vm.passwordInput}}" />
                    <button type="submit" class="btn btn-default">Login <i class="glyphicon glyphicon-log-in"></i>
                    </button>

                </form>
            </div>
        </div>
    </div>
</div>
```

Slika 5.2: `Pogled`, ki je zadolžen za prijavo

Značka `ng-model` omogoča, da povežemo podatke iz kontrole direktno na pogled. Druga značka skrbi, da se ob kliku gumba izvede programska koda iz kontrole. Na sliki 5.3 vidimo kontrolo, ki je vezana na pogled za prijavo iz slike 5.2. V kontroli uporabljamo objekt `$scope`, ki povezuje pogled in kontrolo skupaj. Če pogledamo sliko 5.2 vidimo, da značka `ng-model` vsebuje vrednosti `user.username` in `user.password`. Prav tako imamo v kontroli vezani na pogled prijave, objekt `$scope.user={}`. To pomeni, da se vrednosti, ki se vnesejo v polji za vnos, takoj povežejo s kontrolo, natančneje v objekt `$scope.user`. Ko uporabnik klikne na gumb za prijavo, se preko značke `ng-submit` izvede funkcija `vm.loginUser()`. V funkciji so podani podatki iz objekta `$scope.user`. Ker smo omenili, da kontrola po pravilih dobre prakse ne sme izvajati preveč logike, funkcija `vm.loginUser()` pošlje podatke v procesiranje drugam. To stori s klicem storitve. Storitev je na sliki 5.3 predstavljena kot objekt `AppSrvc`. Vsebino storitve `AppSrvc` lahko vidimo na sliki 5.4. Omenjena storitev ima dve funkciji:

- funkcijo `login`, ki skrbi za prijavo uporabnika v spletno stran
- funckijo `logout`, ki skrbi za odjavo uporabnika iz spletne strani

```
(function() {
  'use strict';
  angular
    .module('stalker.main.controller', [])
    .controller('AppCtrl', ['Session','$rootScope','$scope','$location','AppSrvc',function(Session,$rootScope,$scope,$location,AppSrvc) {
      $scope.vm = {};
      $scope.vm.usernameInput = 'Stalker username';
      $scope.vm.passwordInput = 'Stalker password';
      $scope.user = {};
      $scope.vm.loginUser = function(userCredentials) {
        AppSrvc.login(userCredentials).then(
          function(res){
            if(res){
              var uId = Session.get('user','user_id');
              $location.path("/home/"+uId);
            }
          },
          function(err){
          }
        );
      };
    }]);

})();
```

Slika 5.3: Kontrola, ki je vezana na pogled za prijavo

Ko kontrola `AppCtrl` kliče funkcijo `login` v storitvi `AppSrvc`, se zgodi, da funkcija `login` pošlje `obljubo` kontroli, v kateri obljublja, da bo podatke vrnila takoj, ko bo to lahko storila.

Funkcija najprej preko klica API `login` pošlje podatke zalednemu podsistemu, ki jih je uporabnik vnesel. Ko zaledni podsistem podatke preveri, vrne vrednost `true` ali `false`. V prvem primeru pomeni, da so podatki pravilni, v drugem pa, da so napačni. Ko dobi funkcija v storitvi podatke iz zalednega podsistema, jih posreduje storitvi `Session`, ki shrani podatke v lokalno shrambo brskalnika. Potem izpolni obljubo in vrne kontroli `AppCtrl` podatke, ki jih je obljubila. Če so podatki pravilni, nas podsistem prijavi in preusmeri na osnovno stran `/home`.

```
(function() {
  'use strict';
  angular
    .module('stalker.main.service', [])
    .factory('AppSrvc', ['$location','Session','$q','$http', function($location,Session,$q,$http) {
      var AppService = {};
      AppService.login = function (userCredentials) {
        var deferred = $q.defer();
        var req = {
               method: 'POST',
               url: '/login',
               headers: {
                 'username': userCredentials.username,
                 'password': userCredentials.password
               },
        };
        var success  = function(res){
          if(res.data.length){
            var user = {
              username:userCredentials.username,
              token: res.data[0],
              user_id:res.data[1]
            };
            Session.create(user);
            deferred.resolve(true);
          }else{
            deferred.resolve(false);
          }
        };
        var error = function(error){
          deferred.reject(error);
        };
        $http(req)
        .success(success)
        .error(error);
        return deferred.promise;
      };
      AppService.logout = function () {
        Session.destroy();
      };
      return AppService;
    }]);
})();
```

Slika 5.4: Storitev, ki skrbi za preverjanje pravilnosti vnesenih podatkov pri prijavi

5.2 Osnovna stran

Na osnovni strani pridobimo podatke o vseh vadbah, zabeleženih v podatkovni bazi MongoDB. Na sliki 5.5 vidimo kodo za pogled `osnovna stran`. Za boljšo predstavo lahko v poglavju 6 na sliki 6.3 vidimo zaslonski izgled, ki

ga ustvarimo s kodo `HTML` na sliki 5.5. Slednja vsebuje tri značke AngularJS, ki jih še nismo opisali:

- ng-include
- ng-repeat
- ng-if

Prva značka nam omogoča, da v element HTML, kjer se značka nahaja, prikažemo elemente HTML, ki so zapisani v datoteki, ki jo navedemo kot argument znački `ng-include`. Druga značka `ng-repeat` deluje kot zanka `for`. Omogoča nam, da izluščimo posamezni element iz strukture. Podatki navedeni kot argument v znački `ng-repeat`, so povezani s kontrolo, ki je vezana na pogled `osnovna stran`. V našem primeru je to kontrola `HomeCtrl`. Zadnja značka je `ng-if`. Z njo lahko ustvarimo pogoj, ob katerem se bo

```
<div>
    <div class="">
        <div class="row">
            <div class="col-lg-1 col-md-1 col-sm-1 col-xs-1">
                <div ng-include src="'js/main/main.sideNav.html'"></div>
            </div>
            <div class="col-lg-9 home-content" id="map-canvas" >
                <div class="row">
                    <a ng-repeat="t in test" class="col-lg-4 col-md-5 col-sm-5 col-xs-5" href="#/practice/{{t._id}}">
                        <h1>{{getDate($index)}}</h1><br>
                    </a>
                    <p ng-if="checkResults()" style="margin-left:50px;margin-top:10px;"> Sorry there are no activities recorder from you.
                    </p>
                </div>
            </div>
        </div>
    </div>
</div>
```

Slika 5.5: Koda HTML za pogled `osnovna stran`

značka prikazovala. V našem primeru se bo ta pogoj izpolnil, ko bo funkcija `checkResults()` iz kontrole `HomeCtrl` vrnila vrednost `true`.

Včasih se zgodi, da potrebujemo podatke iz kontrole v standardnih značkah HTML/HTML5. Takrat uporabimo dvojne zavite oklepaje {{}}. Tak primer imamo na sliki 5.5 v znački `href` v argumentu `#/practice/{{t._id}}`. V URL omogočamo dinamično spreminjanje zadnje vrednosti, ki se navezuje na identifikacijsko število neke vadbe. V znački `ng-repeat` izluščimo element `t` iz tabele `test`. Ta element je objekt, ki hrani vrednost posamezne vadbe iz zalednega podsistema. Na sliki 5.6 lahko vidimo programsko

```
(function() {
  'use strict';
  angular
    .module('stalker.home.controller', [])
    .controller('HomeCtrl', ['$rootScope','$scope','$location','AppSrvc','$document','practice',
      function($rootScope,$scope,$location,AppSrvc,$document,practice) {
      $scope.test = [];
      $scope.test = practice;
    $scope.checkResults = function(){
      if($scope.test.length == 0){
        return true;
      }else{
        return false;
      }
    };
    $scope.getDate = function ( index ) {
      var value = practice[index]['practice_date'];
      var date = new Date(value);
      return date.toString();
    }
    }]);
})();
```

Slika 5.6: Programska koda kontrole `HomeCtrl`

kodo kontrole `HomeCtrl`, ki se navezuje na pogled, na katerem smo ravnokar opisovali delovanje značk. Kot lahko vidimo, je spremenljivka `$scope.test` sprva prazna tabela. Nato ji dodelimo vrednost `practice` in jo uvozimo, še preden začnemo izvajati vso logiko kontrole. Vrednost `practice` dobimo, še preden se kontrola začne izvajati, to pa se zgodi v nastavitvi pogleda, kot je predstavljena na sliki 5.7.

```
(function() {
  'use strict';
  angular
    .module('stalker.home.module', [
      'stalker.home.service',
      'stalker.home.controller'
    ])
    .config(['$routeProvider', function ($routeProvider) {
      $routeProvider
        .when('/home/:id', {
          templateUrl: 'js/home/home.index.html',
          controller: 'HomeCtrl',
          resolve: {
            practice:
            ['HomeService','$routeParams',
              function(HomeService,$routeParams){
              return HomeService.getAllPractice($routeParams.id)
              .then(
                function(result){
                  return result.data;
                },
                function(error){
                  return [];
                }
              );
            }]
          }
        });
    }]);
})();
```

Slika 5.7: Nastavitev za pogled `/home`

Nastavitev pogleda velja le, ko bo pot brskalnika URL enaka pogledu `/home`. Tukaj je treba poudariti tri lastnosti:

- v `templateUrl`, v kateri podamo pot do datoteke, je zapisana koda HTML, kako naj izgleda pogled `/home`
- `controller`, v kateri podamo ime kontrole, bo vezana na pogled `/home`
- v `resolve`, se izvede poizvedba po podatkih o vadbah

V lastnosti `resolve` kličemo storitev `HomeService`, ki ima funkcijo `getAllPractice()`. Na sliki 5.8 vidimo programsko kodo za funkcijo `getAllPractice`. Namen te funkcije je, da pridobi iz zalednega podsistema vse vadbe, ki so bile zabeležene.

```
HomeService.getAllPractice = function (id) {
  var deferred = $q.defer();
  var req = {
       method: 'GET',
       url: '/getAllPractice',
       headers: {
       }
      };
  $http(req)
  .success(function(res){
    deferred.resolve(res);
  })
  .error(function(error){
    deferred.reject(error);
  });
  return deferred.promise;
};
```

Slika 5.8: Programska koda funkcije `getAllPractice()` v storitvi `HomeService`

Funkcija `getAllPractice` deluje tako, da preko klica API `getAllPractice` zahteva vse podatke iz zalednega podsistema. Če zaledni podsistem vrne podatke, funkcija izpolni obljubo po vrnitvi podatkov.

5.3 Stran z vsebino vadbe

Na strani z vsebino posamične vadbe prikazujemo:

- čas vadbe
- največjo hitrost med vadbo
- povprečno hitrost cele vadbe
- število kalorij, porabljenih med vadbo
- število kilometrov, ki smo jih prehodili/pretekli
- število korakov

- zemljevid, na katerem so označene točke, kjer smo se nahajali v času vadbe

- graf vibracij med samo vadbo

Vse naštete podatke, razen podatka o kalorijah in vibracijah, smo dobili že na pogledu `/home` ob klicu funkcije `getAllPractice()`. Zato tukaj predstavljamo le delovanje zemljevida in grafa.

```
var data = {
labels: (function (){
  var d = [],pre1 = Math.floor(vibrations[0].x_axis.length / 60);
  if( pre1 == 0){ pre1 = 10; }
  for(var i =0; i< vrednostGrafa;i+=pre1){ d.push(i); }
  return d;
})(),
datasets : [{
    fillColor : "rgba(120,220,220,0.5)",
    strokeColor : "rgba(120,220,220,1)",
    pointColor : "rgba(0,220,220,1)",
    pointStrokeColor : "#fff",
    data : (function(){
      var tab = [],koncali=false,skupajVrednost=0,pomoc=0;
      for(var i = 0;i<vibrations[0].x_axis.length;i++){
        if((i % Math.floor(vrednostGrafa/60)) == 0 && i!=0){
          koncali = true;
          tab.push(skupajVrednost/Math.floor(vrednostGrafa/60));
          skupajVrednost=0;
          pomoc =0;
        }else {
          pomoc = i+1; Math.
          koncali = false;
          skupajVrednost+=Math.sqrt(
            Math.pow(Math.abs(vibrations[0].x_axis[i]),2)+
            Math.pow(Math.abs(vibrations[0].y_axis[i]),2)+
            Math.pow(Math.abs(vibrations[0].z_axis[i]),2)
            );
        }
      }
```

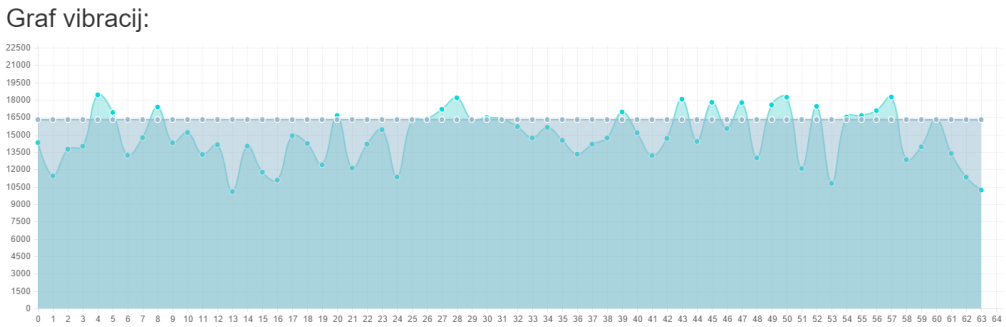
Slika 5.9: Programska koda za nastavitev podatkov v grafu

Na sliki 5.9 je predstavljena programska koda, s katero smo pripravili podatke za graf. Za boljšo predstavo je prikazana z zaslonskim izgledom, ki ga ustvari koda `HTML` iz omenjene slike v poglavju 6 na sliki 6.4.

Podatke smo shranili v objekt `data`, ki ga posredujemo pogledu za prikaz vadbe. Vse podatke smo že dobili takoj, ko smo kliknili na izbrano vadbo. Ti podatki se nahajajo v spremenljivki `vibrations`, ki vsebuje naslednje podatke:

- tabela pospeškov po osi x
- tabela pospeškov po osi y
- tabela pospeškov po osi z
- datum vadbe

Najprej smo v lastnost omenjenega objekta nastavili vrednost `labels`. Ta vrednost predstavlja število vzorcev, ki se jih prikaže na grafu. Te vrednosti smo dodali tako, da smo število vseh podatkov v eni tabeli delili s specifičnim številom v našem primeru s številom 60, ker smo s tem lahko prikazali bolj zglajene grafe. Podrobnejši opis vsebine grafa s slike 5.10 smo opisali v



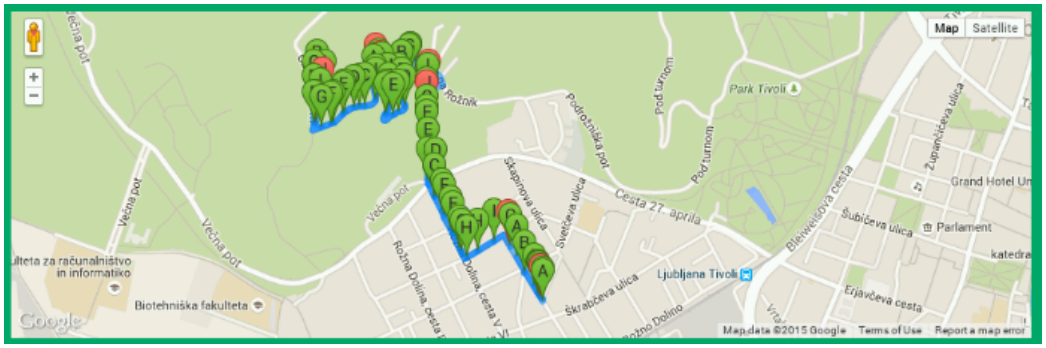
Slika 5.10: Primer grafa teka na Rožnik

poglavju 6, natančneje v podpoglavju 6.2. V drugi lastnosti `datasets` smo dejansko urejali podatke, ki se bodo predstavljali na grafu. Lastnost prejme tabelo objektov in vsak objekt ima podatke o enem grafu. Mi smo v to vrednost dali dva:

- prvega s podatki o povprečju, ki je predstavljal izhodišče
- drugega s podatki o naši vadbi

Prve štiri lastnosti objekta, ki ga dodajamo v tabelo `datasets`, predstavljajo podatke o barvah, ki jih bo imel graf. Podatke vibracij o vadbi smo razdelili na 60 vzorcev. Vsak vzorec hrani povprečje nekega dinamičnega števila podatkov. Ta podatek je namreč odvisen od dolžine naše vadbe. Če smo vadili 20 minut, pomeni to 500 zapisov. Če to vrednost delimo s 60, dobimo število podatkov na vzorec. Glede na omenjen podatek se zapisi seštevajo. Sešteva se jih preko izraza 4.1 v poglavju 4.

Drugi graf, ki je naše izhodišče, generiramo tako, da dodajamo le eno specifično vrednost. V našem primeru je ta vrednost 16927. Ta podatek ni naključen, pač pa predstavlja povprečno vrednost meritve teka s časom 13 minut.



Slika 5.11: Primer zemljevida teka na Rožnik

Pri implementaciji zemljevida je prišlo do težav. Zaradi velikega števila koordinat namreč nismo uspeli prikazati vseh lokacij. Za ta namen smo podatke o lokaciji zmanjšali na 70. Razlog, da smo morali to narediti, je ta, da `Google Maps Engine` ne dovoli več kot 200 oznak lokacij na zemljevidu. Na začetku je kazalo, da bi lahko prikazovali le 10 lokacij za eno pot zaradi omejitve `Google Maps Engine` . Da smo se rešili te zagate, smo na zemljevid postavili več `poti`, ki so se nadaljevale s pričetkom z zadnje lokacije prejšnje poti.

```
navigator.geolocation.getCurrentPosition(function(position) {
  var pos = new google.maps.LatLng(position.coords.latitude, position.coords.longitude);
  $scope.map.setCenter(pos);
  var maxWayPoints = 10,numberOfCalcRoutes = practice[0].locations.length/maxWayPoints;
  for(var i = 0;i<numberOfCalcRoutes;i++){
    directionsDisplay.push(new google.maps.DirectionsRenderer());
    directionsService.push(new google.maps.DirectionsService());
    directionsDisplay[i].setMap($scope.map);
    var tempLocations = [],startTable = i == 0 ? i*maxWayPoints : i*maxWayPoints-1,
    endTable = startTable+(maxWayPoints-1);
    if((practice[0].locations.length-1)-endTable<0){
      endTable = endTable- Math.abs((practice[0].locations.length-1)-endTable);
    } endTable++;
    for(var j = startTable;j <endTable;j++){
      tempLocations.push({location:practice[0].locations[j],stopover:true});
    }
    calcRoute(directionsDisplay[i],directionsService[i],tempLocations);
  }
  function calcRoute(display,service, locations) {
    var start =  locations.shift().location,end =locations.pop().location;
    var tableL = practice[0].locations.length,request = {origin:start,
      destination: end,waypoints: locations,optimizeWaypoints: true,
      travelMode: google.maps.TravelMode.WALKING
    };
     service.route(request, function(response, status) {
      if (status == google.maps.DirectionsStatus.OK) {
        display.setDirections(response);
      }
```

Slika 5.12: Programska koda za nastavitev podatkov v grafu

Na sliki 5.12 je mogoče videti programsko kodo, ki ureja podatke za prikaz na zemljevidu. Najprej nam funkcija `getCurrentPosition` objekta `navigator.geolocation` postavi fokus zemljevida na točko, kjer se nahajamo. Nato delimo število 70 lokacij s številom 10, ker je to največ možnih točk na eno pot. S tem izračunom dobimo število poti, ki jih moramo ustvariti. Vsaki novi poti dodelimo lokacije, ki jim pripadajo. To pomeni, da se lokacije četrte poti nadaljujejo od tam, kjer so se lokacije tretje poti končale. Nato za vsako pot pokličemo funkcijo `calcRoute`, ki poskrbi, da pošlje podatke o poti na `Google Maps Service`. Ta storitev izračuna in nariše pot

na zemljevidu.

Za izračun porabljenih kalorij smo potrebovali:

- vrednost MET, ki predstavlja vrednost za neko aktivnost (mi smo uporabili vrednost 7,5)
- težo v kilogramih
- čas trajanja vadbe podan v urah

Za izračun kalorij smo uporabili izraz 5.1.

$$kalorije = MET * teza\_KG * trajanjeVUrah \tag{5.1}$$

MET vrednost, je vrednost, ki je določena za vsako aktivnost. V našem primeru je to 7,5. Vrednost `teza_KG` vsebuje težo osebe, ki je izvajala vadbo in zadnja vrednost `trajanjeVUrah` je čas trajanja vadbe podan v urah [14].

Poglavje 6

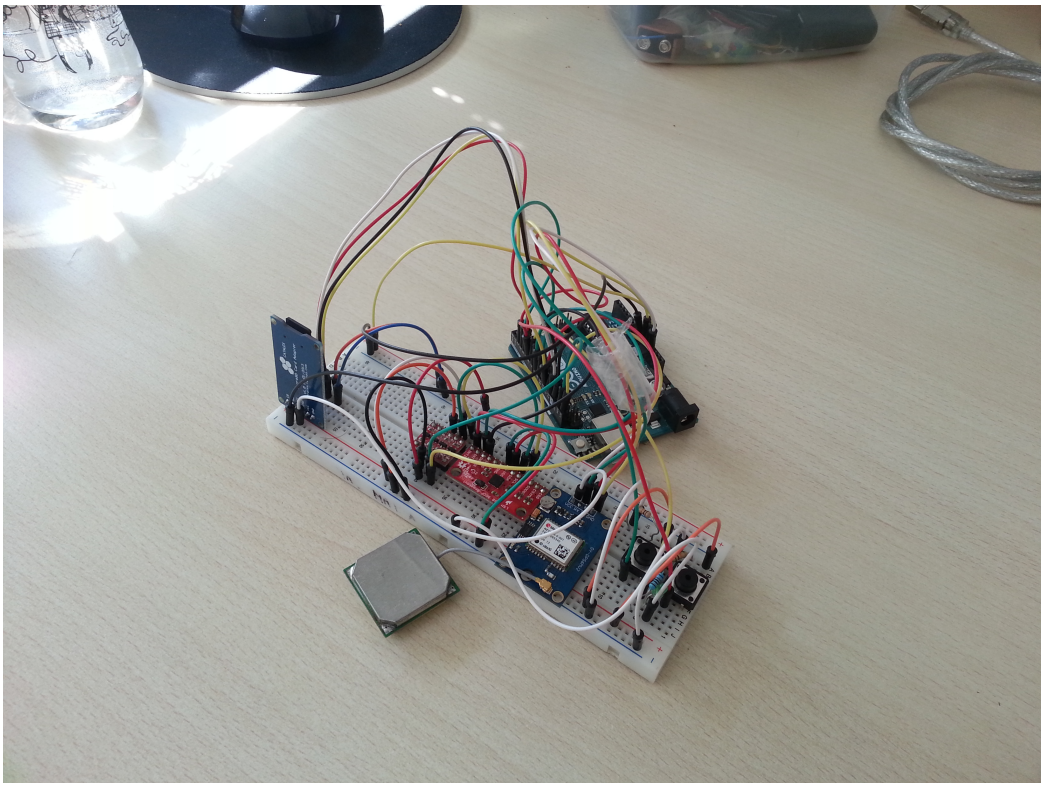
Delovanje celotnega sistema

Celoten sistem sestavljata prenosni podsistem in zaledni podsistem. Ideja pri razvoju sistema je bila že od začetka ločena na dva dela. Prvi je zajem podatkov, drugi pa prikaz podatkov. Ker sta najbolj splošni obliki rekreacije hoja ali tek, je uporabnost učinkovita predvsem v omenjenih primerih rekreacije.

6.1 Opis uporabe

Prenosni podsistem priključimo na baterijo in ga spravimo v torbico za okrog pasu. Naprava se avtomatsko zažene in prične iskati satelite. Ko naprava vzpostavi povezavo z ustreznim številom satelitov, se LED-dioda prižge in tako lahko pritisnemo tipko, namenjeno za začetek vadbe. Naprava ne bo shranjevala podatkov, če povezava GPS ne bo delovala. Nato lahko začnemo z vadbo, ki je priporočljiva predvsem na odprtem oz. kjer ni velikih zgradb in preveč dreves.

Po opravljeni vadbi je treba pritisniti gumb za zaključek vadbe. S tem se naprava postavi v stanje, kjer pričakuje vzpostavitev povezave z zalednim podsistemom. Prenosni podsistem in zaledni podsistem povežemo med seboj s kablom USB. Preden se začnejo podatki prenašati, je treba počakati 20 sekund. Ko se prenos začne, se vključi rdeča LED-dioda. Trajanje prenosa



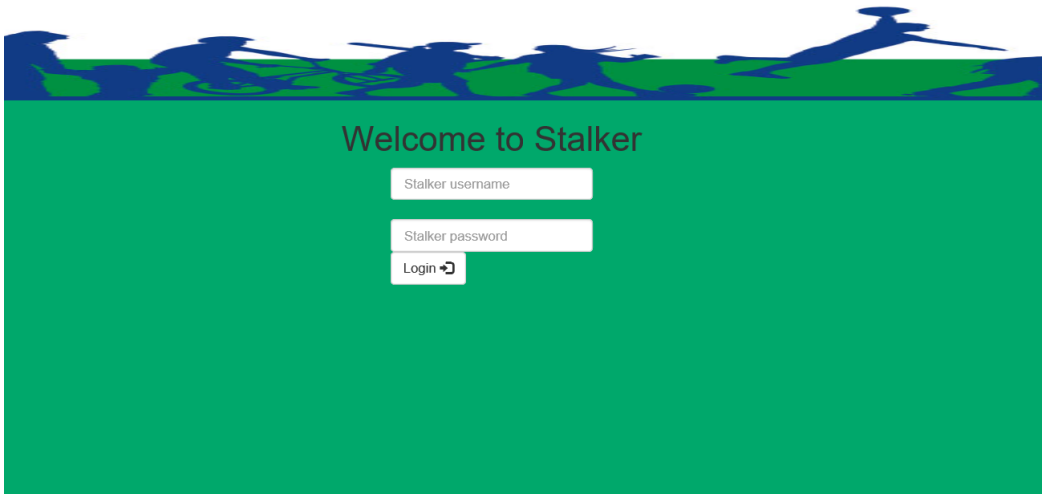
Slika 6.1: Izgled prenosnega podsistema

med napravama je odvisno od števila podatkov (npr. za 10 minut vadbe je trajal prenos podatkov 30 sekund). Daljša kot je vadba, daljši je prenos med napravama. Ko se prenos zaključi, se vključijo vse LED-diode. Takrat se samodejno pobrišejo tudi vsi podatki na spominski kartici na napravi, naprava pa je pripravljena na novo vadbo.

Če želimo preveriti podrobnosti teka, se moramo prijaviti na spletni strani, ki jo gosti strežnik. To storimo z uporabniškim imenom in geslom. Izgled prijavne strani je viden na sliki 6.2.

Če smo vnesli pravilne podatke pri prijavi, nas spletna stran usmeri na drug pogled, kjer so razporejene vse vadbe z datumi in časom posamezne vadbe. To je vidno na sliki 6.3.

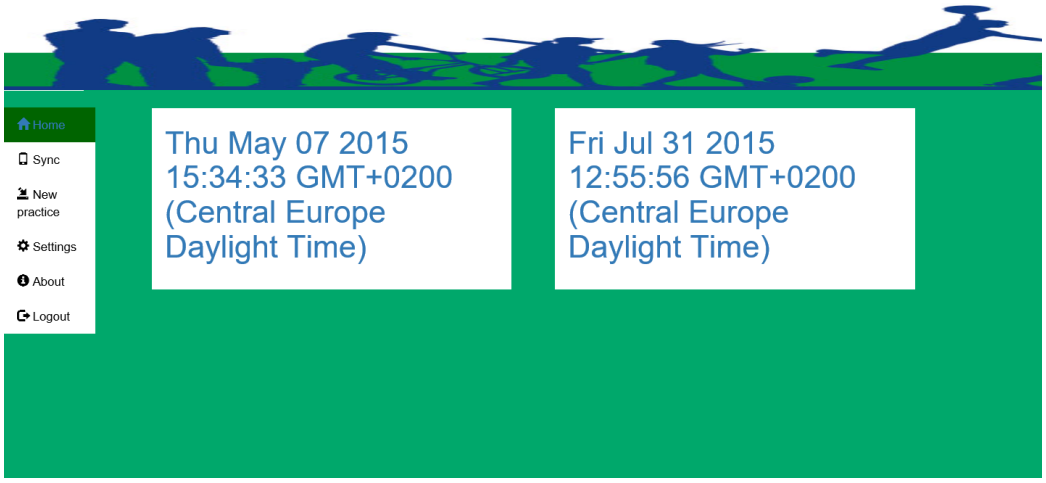
Za dostop do posamezne vadbe je treba klikniti na posamezni kvadratek, ki predstavlja posamezno vadbo. Ko to storimo, nas spletna stran usmeri



Slika 6.2: Prijavna stran na spletni strani

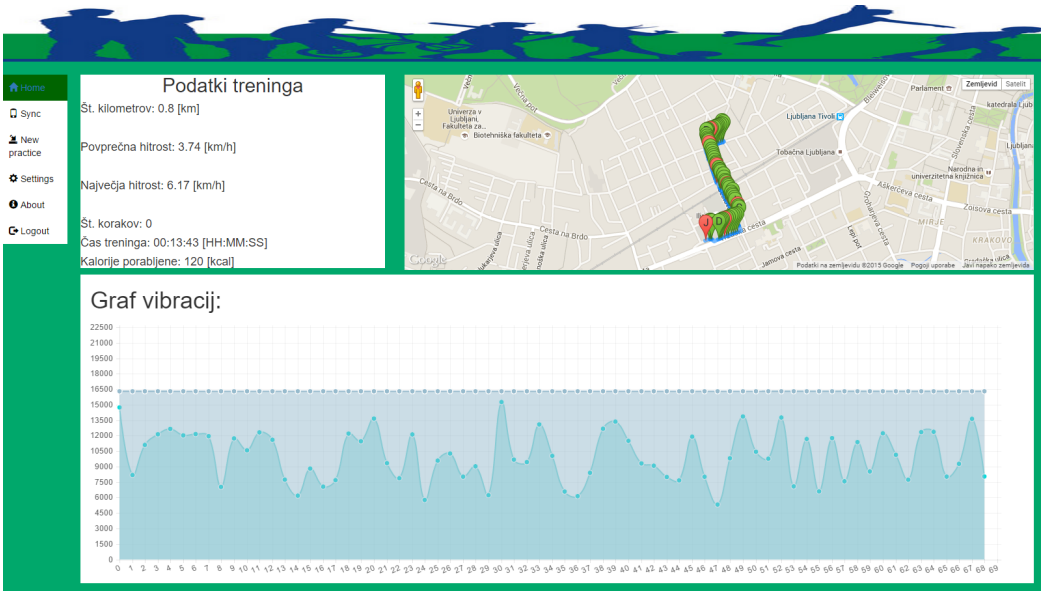
na pogled, ki vsebuje podrobnosti izbrane vadbe. Na tem pogledu, kot je razvidno iz slike 6.4, vidimo:

- zemljevid z lokacijami, kjer smo se nahajali med vadbo
- graf vibracij
- največjo hitrost med vadbo



Slika 6.3: Osnovna stran spletne strani

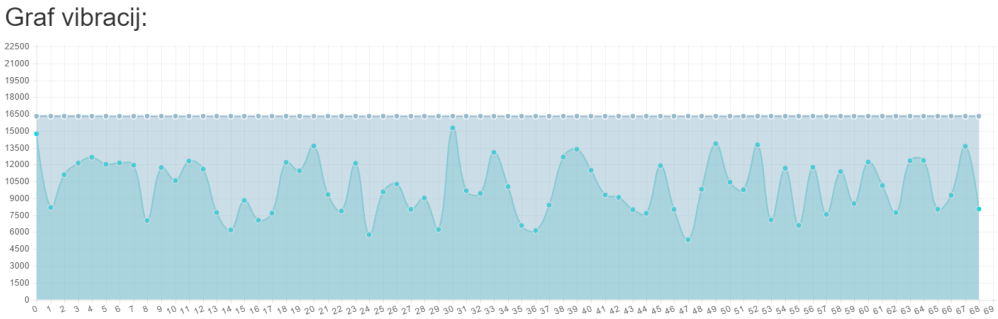
- povprečno hitrost vadbe
- število kilometrov
- število korakov
- porabo kalorij
- čas vadbe



Slika 6.4: Pogled z vsebino izbrane vadbe

6.2 Analiza vibracij

Vibracije, ki se prikazujejo na grafih, se razlikujejo od osebe do osebe. Za oporno točko smo vzeli meritve ene osebe. Za čim bolj natančne podatke bi bilo treba opraviti več testov, pri tem pa upoštevati fizične lastnosti posamezne osebe(teža, višina). Za boljšo primerjavo bi bilo dobro opraviti vadbe na različnih položnih podlagah (npr. na položni gozdni poti in položni asfaltirani podlagi. Tako bi dobili boljšo analizo obremenitve telesa med vadbo na posamezni podlagi.



Slika 6.5: Graf, ki prikazuje podatke med hojo

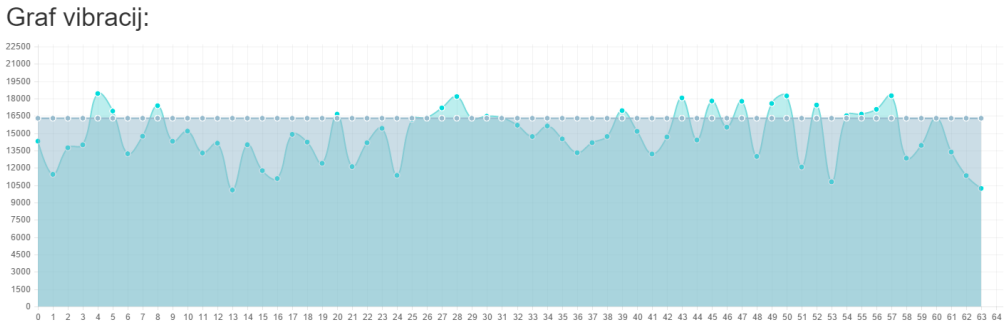
Poleg podatkov o hitrosti, pretečeni razdalji, porabljenih kalorijah in času vadbe, imamo tudi graf vibracij. Na slednjega smo se bolj osredotočili, ker potrebuje več analiz.

Za namen analize smo naredili tri teste:

- enega med hojo
- drugega med tekom v klanec
- tretjega med tekom po mestu

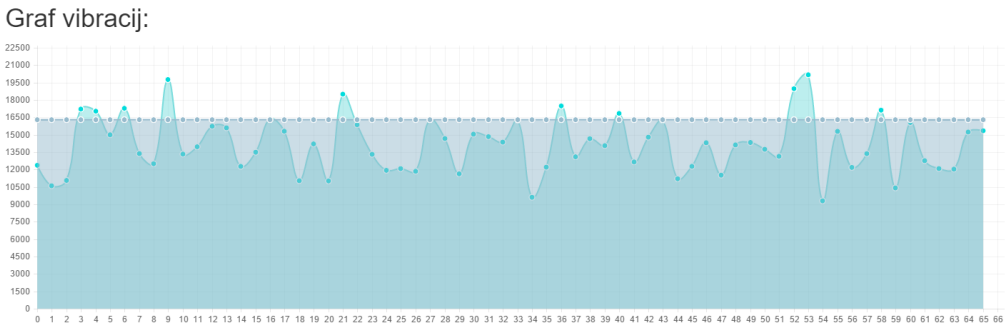
Grafi na slikah 6.5, 6.6 in 6.7 prikazujejo vrednosti vibracij med določeno vrsto rekreacije. Na osi y so prikazane digitalne vrednosti, ki so pretvorjene iz analognih. Te vrednosti se gibljejo od vrednosti 0 in se povečujejo za interval 1500 do vrednosti 22500. Na osi x pa imamo vrednosti od 0 do 64. Posamezna vrednost na osi x predstavlja vzorec meritve. Če imamo npr. 255 zapisov o vibracijah. Potem predstavlja en vzorec povprečje vrednosti 3 zapisov vibracij. Število zapisov na vzorec je odvisno od celotnega števila zapisov vibracij. Število zapisov na vzorec dobimo z enačbo $stZapisovNaVzorec = vsiZapisi/65$. Vsi trije grafi imajo za izhodiščno točko povprečje, izmerjenega med tekom po mestu.

Na sliki 6.5 je mogoče videti graf podatkov med hojo. Vidi se, da se podatki med samo hojo ne morejo niti v največjem primeru približati povprečju med tekom.



Slika 6.6: Graf, ki prikazuje podatke med tekom v klanec

Sliki 6.6 in 6.7 prikazujeta grafa podatkov med tekom v klanec in tekom po mestu. Pri teku v klanec moramo omeniti, da je tek potekal na začetku in na koncu po asfaltirani podlagi medtem, ostalo pa po gozdni poti. Zato sta prva in zadnja sedmina grafa nepomembni. Vidimo lahko, da pri teku v klanec na sliki 6.6 imamo od tretje pa do šeste sedmine grafa višje meritve v primerjavi z grafom teka po mestu na sliki 6.7. Razlika nam pove, da pri teku v klanec so prisotne večje vibracije. Še bolj očitna razlika je med peto in šesto sedmino grafa, kjer so meritve teka v klanec navzdol. Opazno je, da so meritve višje od meritev grafa med tretjo in peto sedmino, kjer smo tekli v klanec. Zato bi lahko ocenili, da tek po klancu navzdol obremenjuje bolj telo kot tek v klanec. Pri teku v klanec je potrebno upoštevati, da je



Slika 6.7: Graf, ki prikazuje podatke med tekom po mestu

tek počasnejši kot tek po mestu. To bi lahko bil tudi razlog zakaj so meritve

višje. V vsakem primeru tek v klanec bolj obremeni telo v smislu vibracij.

Da bi lahko oblikovali bolj konkretne trditve. bi bila potrebna obširnejša testiranja. Podatki lahko po drugi strani varirajo zaradi utrujenosti tekača in splošnega počutja te osebe.

Poglavje 7

Sklepne ugotovitve

Po razvoju celotnega sistema smo ugotovili, da je veliko možnosti tudi za nadaljnji razvoj. Prenosni podsistem lahko naredimo še manjši z izdelavo svojega vezja. Funkcije kot sta štetje korakov, prikaz vibracij so lahko s testiranjem in analizo natančnejše.

Pri razvoju celotnega sistema je prišlo do problemov, ki vplivajo na samo delovanje. Prvi je slaba avtonomija baterije. Z novo 9 V baterijo lahko opravimo le do 20 minut vadbe, zato je treba za boljšo avtonomijo vzporedno povezati več 9 V baterij. Ena izmed rešitev bi bila uporaba baterije za modelarje, vendar so problem teh baterij zelo dragi polnilci. Zaradi finančnih omejitev se ta rešitev ni realizirala, pač pa se je realizirala rešitev z vzporedno vezavo baterij.

Drugi problem, ki se nanaša na nadgradnjo sistema, je število prostih priključkov na prenosnem podsistemu. Sedaj so namreč praktično vsi digitalni vhodi/izhodi zasedeni. Za nadgradnjo kakšnih drugih tipal bi bilo treba razmisliti o drugi napravi Arduino ali alternativi napravi Arduino. Pri tem problemu je treba omeniti dejstvo, da trenutna programska koda, ki se izvaja na prenosnem podsistemu, zavzame skoraj 90 % prostora. Tudi v primeru, da ne bi dodajali novih tipal, bi z razširitvijo programske kode povzročili, da bi imel prenosni podsistem premalo prostora za hranjenje vseh programskih kod.

Opazili smo tudi, da modul GPS občasno zgreši lokacijo, predvsem ko smo pod streho ali pod visoko zgradbo, ki ovira signal.

V diplomskem delu smo si prizadevali, da bi implementirali funkcionalnost štetja korakov. Funkcionalnost je vključena v prenosni podsistem, ampak zaradi pomanjkanja testov še ni dovolj natančna.

Prvi korak pri nadaljnem razvoju so testi, da ugotovimo koliko kvalitetne informacije pridobimo preko vibracij in da do konca realiziramo štetje korakov. Takoj za tem bi se osredotočili na izdelavo svojega vezja, da bi bil prenosni podsistem bolj priročen. Obenem nameravamo začeti razvijati podsistem, ki bo omogočal priključitev več različnih prenosnih podsistemov na skupni zaledni podsistem. S tem podsistemom bi omogočili večji skupini ljudi, uporabo istega zalednega podsistema, znotraj iste skupnosti.

Literatura

[1] Arduino IDE. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Arduino. [Dostopano 29. 08. 2015].

[2] AngularJS. [Online]. Dosegljivo: https://angularjs.org/. [Dostopano 29. 08. 2015].

[3] ExpressJS. [Online]. Dosegljivo: https://http://expressjs.com/. [Dostopano 29.08.2015].

[4] MongoDB. [Online]. Dosegljivo: http://docs.mongodb.org/manual/. [Dostopano 29. 08. 2015].

[5] NodeJS. [Online]. Dosegljivo: https://nodejs.org/about/. [Dostopano 29. 08. 2015].

[6] Raspbian OS. [Online]. Dosegljivo: https://www.raspbian.org/. [Dostopano 29. 08. 2015].

[7] Debian OS. [Online]. Dosegljivo: https://www.debian.org/. [Dostopano 29. 08. 2015].

[8] Processing IDE. [Online]. Dosegljivo: https://processing.org/reference/environment/. [Dostopano 29. 08. 2015].

[9] Knjižnica node-forge. [Online]. Dosegljivo: https://www.npmjs.com/package/node-forge. [Dostopano 29. 08. 2015].

[10] Visual Studio Code. [Online]. Dosegljivo: https://code.visualstudio.com/. [Dostopano 29. 08. 2015].

[11] Knjižnica za GPS modul. [Online]. Dosegljivo: http://arduiniana.org/libraries/tinygps/. [Dostopano 28. 08. 2015].

[12] Knjižnica za pospeškometer. [Online]. Dosegljivo: https://github.com/sparkfun/SparkFun_LSM9DS0_Arduino_Library. [Dostopano 28. 08. 2015].

[13] Global positioning system:Basic concept of GPS. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Global_Positioning_System#Basic_concept_of_GPS. [Dostopano 28. 08. 2015].

[14] Running Calorie Calculator [Online]. Dosegljivo: http://www.freeonlinecalculatoruse.com/runningcaloriecalculator.html. [Dostopano 28.08.2015].

[15] GNU General Public Licence. [Online]. Dosegljivo: https://www.gnu.org/copyleft/gpl.html. [Dostopano 20. 9. 2014].

[16] Raspberry Pi 2 Model B [Online]. Dosegljivo: http://core0.staticworld.net/images/article/2015/02/raspberry-pi-2-sd-card-100569129-orig.png. [Dostopano 28.08.2015]

[17] Arduino Uno R3 [Online]. Dosegljivo: https://cdn.sparkfun.com//assets/parts/6/3/4/3/11021b-00.jpg. [Dostopano 28.08.2015]

[18] Tabela specifikacij različnih Arduino ploščic [Online]. Dosegljivo: https://www.arduino.cc/en/Products.Compare. [Dostopano 28.08.2015]

[19] Arduino Lilypad ploščica [Online]. Dosegljivo: https://www.arduino.cc/en/uploads/Main/LilyPad_5.jpg. [Dostopano 28.08.2015]

[20] Arduino Nano ploščica [Online]. Dosegljivo: https://www.arduino.cc/en/uploads/Main/ArduinoNanoFront_3_sm.jpg. [Dostopano 28.08.2015]

[21] GPS modul [Online]. Dosegljivo: https://www.arduino.cc/en/Products.Compare. [Dostopano 28.08.2015]

[22] Pospeškometer, giroskop, barometer [Online]. Dosegljivo: https://cdn.sparkfun.com//assets/parts/9/3/1/9/1263601.jpg. [Dostopano 28.08.2015]

[23] Pretvornik napetosti [Online]. Dosegljivo: https://cdn.sparkfun.com//assets/parts/8/5/2/2/1200906.jpg. [Dostopano 28.08.2015]

[24] Modul za spominsko kartico microSD [Online]. Dosegljivo: https://d3s5r33r268y59.cloudfront.net/27271/products/thumbs/2014-05-16T16:00:29.024Z-237.jpg.855x570_q85_pad_rcrop.jpg. [Dostopano 28.08.2015]

[25] Vezalna shema med pospeškometrom in Arduino napravo [Online]. Dosegljivo: https://cdn.sparkfun.com/r/600600/assets/e/2/8/f/f/advanced-redboard_levelShift_bb.png. [Dostopano 28.08.2015]

[26] Vezalna shema med napravo Arduino in modulom GPS [Online]. Dosegljivo: http://thefortuneplanet.com/wpcontent/uploads/2014/11/ArduinoGPS_bb-1024x600.png. [Dostopano 28.08.2015]

[27] Vezalna shema med modulom za spominsko kartico in Arduino napravo [Online]. Dosegljivo: http://4.bp.blogspot.com/-vygHs1SqzNI/U9zEjUEhQUI/AAAAAAAAKzw/Wv-VH0O9Wq4/s1600/SD+Card+Sketch_bb.jpg. [Dostopano 28.08.2015]